Steve Jenkin
20/3/84

AUUGN

Australian Unix User Group Newsletter

Volume 5
Number 2

Registered by Australia Post Publication No. NBG6524

The Australian UNIX* Users Group Newsletter

Volume 5 Number 2

March 1984

CONTENTS

* UNIX is a trademark of Bell Telephone Laboratories.

Editorial

"Well, thats issue 1 of volume 5 out of the way!" "What is all this stuff still on my desk?". "It must be issue 2 of volume 5!"

In This Issue

This issue contains the overflow from volume 5 number 1, a greater than normal amount of network mail plus information from the February AUUG meeting. Further summaries of talks from the meeting will be published next time. I apologise to our American readers as they have probably already seen the summary of the North American Summer 1983 UNIX meeting. Although this meeting was held some time ago, few Australian readers will have seen a summary.

AUUG Meeting in Sydney

The AUUG meeting held at the University of Sydney in February was a raging success, in more ways than one. More than 250 people attended the meeting and vendor exhibition. I have received many favourable comments and I extend to the organisers the thanks of all attendees. The next meeting will be held in Melbourne on August 27 and 28, 1984. Preliminary announcements will be made in April and the contact person is:

Robert Elz
Department of Computer Science
University of Melbourne
Parkville VIC 3052

(03) 341 5225

Aunet: kre:munnari

The Summer 1985 meeting is tentatively in Sydney or Wollongong.

During the AUUG business session on the second day of the meeting an interim constitution was adopted and the following list of people appointed to an interim Executive Committee.

Name	Organisation	Position
John Lions	University of NSW	Convenor
Geoff Cole	University of Sydney Comp. Centre	Deputy Convenor
Chris Maltby	University of Sydney	Treasurer
Kevin Hill	University of NSW	Secretary
Peter Ivanov	University of NSW	Newsletter Editor
David Horsfall	University of NSW	Committee Member
Doug Richardson	University of Sydney	Committee Member
John Field	CSIRO Adelaide	Committee Member
Robert Elz	University of Melbourne	Committee Member
Ross Nealon	University of Wollongong	Committee Member
Tom Crawley	University of Western Australia	Committee Member
Chris Campbell	Digital Electronics	Committee Member
Phil Chadwick	Queensland Department of Forestry	Committee Member
John O'Brien	Fawnray	Committee Member
Tim Roper	University of Queensland	Committee Member

Fees have not been finalised but should be approximately $10 for membership and $30 for a normal newsletter subscription. Newsletter subscriptions will be an additional $6 (surface mail) or $30 (air mail) for readers outside Australia. Newsletter price changes will be applied to new subscriptions or subscription renewals received after the fees are finalised.

The Executive Committee will report to the next meeting on a final constitution. Members of the executive may be (collectively) mailed through "auugexec:elecvax".

Quote of the Month

Seen in another newsletter recently. An article on yet another C compiler, under a sub heading "Problems Learning C":

> "The first trick I had to learn, which is not emphasised enough in my view, is to read a line of code "inside-out", not "left to right". By inside-out I mean to read the most deeply bracketed expression first and then move out through the brackets."

The hint is fellas, "Bang the rocks together!".

Contributions

Just because I have published two issues in short succession, do not get the idea that I have sufficient material. If you do something, almost anything within reason, write it up and send it to me. I still have a few books to give away to the best contributor to each issue.

Books

Prentice-Hall of Australia and Holt-Saunders have supplied information about four books to be published/released in the near future. Adding to our list, they are

Books on UNIX

15. UNIX System V: A Quick Reference Guide
Wetzel
Prentice-Hall

Books on C

7. Programmer's Guide to C
Lees
Prentice-Hall

8. Programming in C
Traister
Prentice-Hall

9. Programming in C
Stephen G. Kochan
Hayden Book Company
(Australian Distribution through Holt-Saunders Pty Ltd)

I have been unable to obtain a book review in time for this issue, but several are in the pipeline. If you want to review any of the books above, or those in the last issue, drop me a line.

Nets

The network database mentioned in the last newsletter is operational. At present it contains data on 1200 machines (nodes) interconnected by over 3000 links. Typical entries contain the following items of information:

name	The network name of this node, eg "decvax".
group	A code specifying the geographical location of the node. These codes consist of a three letter country abbreviation (usa, eur, can and aus at present) followed by a period and some other group of letters. Unfortunately, who ever selected the other letters has not seen fit to tell us what they mean.
address	The name and (real mail) address of the node contact person.
phone	The telephone number of the node contact person.
netaddress	The network mailing address of the contact person.
news	List of nodes with which this node exchanges network news.
mail	List of nodes with which this node exchanges network mail.

This information, and information on mail routes between nodes, is available to anyone on the AUNET by mailing your query to "auugnetdb:elecvax". Queries will be automatically processed and are made up of one or more requests. The possible requests are

#info node	Returns all information about the network node with name "node".
#match pattern	Returns information on all nodes where "pattern" occurs in the contact "address". The matching process ignores the case of characters. The pattern may contain `?' to match any single character and `*' to match zero or many characters.
#path node1 node2	Produces a list of the shortest network mail paths from node1 to node2.

I am making every effort to keep the database up to date, but should you obtain firm information that contradicts results obtained from "auugnetdb" PLEASE MAIL "peteri:elecvax" or "auugn:elecvax" so that the database can be corrected.

The following AUNET sites are new or have sent updated entries.

==========

Name: csu40
Address:Computing Services Unit
University of NSW
PO Box 1
Kensington NSW 2033
Phone: +61 2 662 3590
Machine:
PDP-11/40, 2*RK05-J, RK05-F, DJ-11, 124Kw core.
Unix level 7 Ausam

Serves as SUN switching node - no user accounts. (Has 10 nodes!)
Contacts:
Dave Horsfall (dave:csu60)

==========

Name: csu60
Address:Computing Services Unit
University of NSW
PO Box 1
Kensington NSW 2033
Phone: +61 2 662 3590
Machine:
PDP-11/60, 2*RK05-J, Ampex DM980 with AED 8000 controller,
TU10, 2*DZ11, LV11, DP11, DR11-B, user control store.
Unix level 7 Ausam

The CSU production machine - takes over from "csu40" except network.
Contacts:
Dave Horsfall (dave:csu60)

==========

Summary of a talk given at the Summer UNIX Meeting by
Bill Murphy
AT&T
on
"How to Obtain UNIX"

Summary by
Peter Ivanov
University of New South Wales

March 9, 1984.

Bill was the first of the "surprise AT&T speakers". Lawrence Brown from AT&T Bell Laboratories spoke on "Development Directions" and his talk will be summarised later. Rather than try to cover everything Bill said I will outline a few of the more interesting points and follow them with the reproduction of a DRAFT letter to be distributed to UNIX licensees in the near future.

As we all know, System V was released last year and System V Release 2 has been announced and is due for shipment starting April 1, 1984. System V deliveries outside the U.S.A. have been delayed because the U.S. Government has taken exception to the password encryption algorithm. The fact that previous UNIX systems have been shipped with the same algorithm does not appear to matter and so steps are being taken to remove the password code from the international distribution. In answer to a question from the audience on "What do we use as a password algorithm" Bill smiled and said "The code in previous versions of UNIX works perfectly well".

If you are dismayed by the fact that you have only just received System V (because of the embargo) and now System V Release 2 is just around the corner, you may be in luck. If your license was processed between December 1, 1983 and April 1, 1984 then you will receive Release 2 at no charge.

Full details of license costs appear in the draft letter, but Educational licensees should note the following. AT&T charges $800 to license as many CPUs as you like in one go. Any CPUs that you later wish to ADD to the list will cost $400. So include every CPU you can find in that first group for $800, even if if will never run UNIX. The reason is that it costs nothing to transfer the license from one CPU to a new one.

What follows is a DRAFT letter from AT&T International, copies of which were provided by Bill Murphy to the Sydney UNIX meeting on February 20, 1984.

DRAFT letter from

AT&T International
Mt. Kemble Ave - Rt 202
P O Box 7000
Basking Ridge NJ 07920

Effective January 1, 1984, divestiture of the Bell System in the United States took place causing a major reorganization and change of the Corporate name. Former Corporate names such as American Telephone and Telegraph Company (AT&T), Western Electric Company and Bell Laboratories, have been replaced with AT&T, AT&T Technologies, Inc. and AT&T Bell Laboratories.

Effective on that date, all the rights and privileges presently granted under Software Licensing Agreements with American Telephone & Telegraph (AT&T) and the Western Electric Company are transferred to, and will become a part of, AT&T Technologies, Inc.

AT&T International will continue as the international representative of products and services for AT&T Technologies, Inc.

On January 18, 1984, at the UniForum Conference in Washington, D.C., AT&T Technologies announced the availability of UNIX* System V, Release 2.0, effective April 1, enhanced Level I support to encompass support for any source license regardless of the machine, more flexible policy on the licensing of Customer Provision Agreement, along with three other add on software applications.

Unix System V, Release 2.0 includes enchancements and performance improvements that build upon improvements offered with System V, Release 1.0 in February, 1983. Some combined improvements are:

- VI Screen Editor
- Job Control Capability
- Interprocess Communications
- 5% to 10% faster then Release 1.0
- Up to 25% performance improvements in the Shell

The three add on software applications are:

- UNIX BASIC Software - UNIX BASIC allows for rapid programming of simple applications in BASIC language for business and scientific purposes.

- UNIX Documenter's** Workbench Software - Documenter's Workbench takes the drudgery out of text processing and helps you format complex text for displays and typsetting, quickly and efficiently.

- MC68000 UNIX Software Generation System - MC68000 Software Generation System is for developing software for a computer system based on the Motorola 68000 micro-processor.

* UNIX is a trademark of AT&T Bell Laboratories

** Documenter Workbench is a trademark of AT&T Technologies, Inc.

Machine Readable Documentation for UNIX System V, Release 2.0 is now available on magnetic tape, under license agreement, for a fee of $10,000. It contains formatting codes for association and use with UNIX Documenter's Workbench Software.

We wish to thank those of you who were able to join us for the Business and Technical Seminar on UNIX Software in Las Vegas, Nevada, in December, 1983. We found the meeting extremely valuable, and judging from the many comments the effort was well received and appreciated by you, our customers.

We've also had an opportunity to discuss the concerns, ideas and suggestions you shared with us at the meeting. As a result, we have changed our policies with regard to:

- the use of the run-time libraries in customer-developed applications software may now be included in customer-developed application software systems without payment of a sub-licensing fee.

Standard C Library	/lib/libc.a
Math Library	/lib/libm.a
Object File Access Library	/lib/libld.a
Fortran Library	/usr/lib/libF77.a

- We have identified the following files related to the spell command which may be included in sub-licensing object code systems.

```
/usr/src/cmd/spell/american
/usr/src/cmd/spell/extra
/usr/arc/cmd/spell/stop
/usr/src/cmd/spell/british
/usr/src/cmd/spell/list
/usr/src/cmd/spell/hashcheck
/usr/src/cmd/spell/hashmake
/usr/src/cmd/spell/local
/usr/src/cmd/spell/htempl
```

Attached are brochures and prices along with information on other UNIX System related applications released during 1983. Orders are being accepted now for shipment after April 1, 1984, of UNIX System V - Release 2.0, UNIX System V BASIC Interpreter, UNIX System V MC68000 Software Generation System and UNIX System V Documenter's Workbench. All of the other applications are available now for shipment, upon processing the software agreements and receipt of payment. These are: UNIX System V Writer's Workbench, UNIX System V Instructional Workbench and the UNIX System Software for the Teletype*** 5620 terminal.

Effective with the ordering of UNIX System V, Release 2.0 we are announcing as an introductory offer the provision of three months of free Level I Basic Support.

*** Teletype is a trademark of AT&T Teletype

Support for UNIX Software is provided by AT&T Technologies, Inc. from its Network Service Center, Lisle, Illinois, U.S.A. Level I Support consists of:

a. Monthly newsletter

b. Update for software and documentation

c. Problem reporting facility

d. Distribution of known problem list

Level I Support normally costs $150.00 a month for a minimum of 12 months.

Level I and II Hot Line Support costing $350.00 per month for a minimum of 12 months is described in the attached brochure. One month of free Level II Support will be provided to new Level I customers.

Customer Provision Agreements and Fees

New discount structures have been developed for sub-licensing under Customer Provisions Agreements for the Licensing of System V, Release 2.0, and the following UNIX System V add on applications: Documenter's Workbench (DWB), MC68000 Software Generation System (SGS) and BASIC Interpreter, Writer's Workbench (WWB) and "S" Statistical Software.

Option 1 - Existing Customer Provision Agreements may continue in effect at the option of the customer.

Option 2 - Existing Customer Provision Agreements may be discontinued and a new Customer Provision Agreement schedule established allowing credit for the previous $25,000 payment.

Option 3 - The previous Customer Provision Agreement may also be continued and the customer may also start with a new Customer Provisions Agreement upon payment of an additional $25,000 fee and both schedules may be used upon the option of the customer as appropriate.

The following sub-licensing fees are now in effect including the up front fees to establish the Customer Provisions Agreement:

Sub Licensing Fee Schedule

	UNIX System V	WWB	DWB	MC6800 SGS	BASIC	"S" STATISTICAL
Up-Front Fee	$25,000	$3,000	$3,000	$10,000	$10,000	$5,000
Binary Sublicensing Fees						
1-2 User System	60	50	10	50	50	100
1-8 User System	125	100	15	100	100	200
1-16 User System	500	150	30	250	250	400
1-32 User System	1000	250	45	500	500	800
1-64 User System	3500	350	125	1500	1000	1400
64+ User System	7000	550	250	2500	1500	2000

Discount from Sub-Licensing Fees & Commitment Options

Discounts of 2% for each $100,000 of sales commitment may be taken in sublicensing fees up to a maximum discount of $3 million or 60%. Discount schedules will be established on an annual basis, payable quarterly, and the customer is eligible to renegotiate the discount schedule upward or downward annually. Commitments are final for the 12-month period and the customer is obligated to pay regardless of whether the revenue objectives are achieved or exceeded.

1. As an introductory offer we encourage you to consider moving to these new schedules by allowing discounts for the first year to be based upon all previous sublicensing fees paid. As an existing customer we will advise you of the accumulated sublicensing fees previously paid from which you can, at your option, apply the new discounts and make payments accordingly.

2. Customers who anticipate higher sales in the next 12 months can increase their commitment and the discount will be based on the committed volume rather than actual sales in the preceding year. However, in this case, one-fourth of the commitment must be paid at the end of each quarter even if the sales were lower than anticipated. In addition, actual sales must still be reported quarterly and any additional payments made 30 days after the reporting period.

To accommodate licensees who require extensive use of source code on multiple CPUs, we are introducing a new Multiple source licenses arrangement for UNIX System V. A commercial licensee having at least two CPU's licensed for source and a Customer Provisions Agreement may obtain licenses for use of source code on additional CPU's at the following fees:

$1000 for a 1-32 user system
$3500 for a 1-64 user system
$7000 for a system with capacity for more than 64 users.

It is suggested that these schedules be discussed with our representative in order to establish a clear understanding of the commitment that would be provided in a written agreement.

License Agreements or modifications to agreements will be drawn up and sent to you based upon mail or telex requests to the address on our letterhead.

Effective with this reorganization our commitment to enhance and upgrade the UNIX Operating System continues. Please let us hear from you as to how we may assist you in your use of the UNIX System. Your suggestions on how we may improve the utilization of the UNIX System by either technical enhancements or marketing support will be appreciated.

Marketing Manager - UNIX Systems

Item	Commercial	Educational
Unix System V, Release 2.0 Source Code (1)	$43,000.00	$800.00
Each Additional CPU	$16,000.00	$400.00
Customer Provisions Supplemental Agreement	$25,000.00	-
Multiple Source Licenses	1-32 Users $1000 1-64 Users $3500 64+ Users $7000	-
Unix System V, Release 2.0 Administrative-Educational	-	$16,000.00
Upgrade Fees from UNIX System V, Release 1.0 (1)(2)	$2,500.00 All CPU's	$800.00 All CPU's
Add on packages (source code only)		
BASIC Interpreter First CPU	$5,000.00	$2,500.00
BASIC Interpreter Additional CPU's	$2,500.00	$1,250.00
Customer Provisions Fee	$10,000.00	-
MC68000 Software Generation System First CPU	$7,500.00	$3,000.00
MC6800 Software Generation System. Additional CPU's	$3,750.00	$1,500.00
Customer Provisions Fee	$10,000.00	-
Documenter's Workbench First CPU	$4,000.00	$1,500.00
Documenter's Workbench Additional CPU's	$2,000.00	$500.00
Customer Provisions Fee	$3,000.00	-

Item	Commercial	Educational
Writers Workbench First CPU	$4,000.00	$2,000.00
Writers Workbench Additional CPU's	$1,600.00	$800.00
Customer Provisions Fee	$3,000.00	-
"S" Statistical Software	$8,000.00	$400.00
"S" Statistical Software Additional CPU's	$3,000.00	$400.00
Customer Provisions Fee	$5,000.00	-
Instructional Workbench (Binary). First CPU	$2,500.00	$1,250.00
Instructional Workbench (Binary). Additional CPU's	$2,500.00	-

(1) Includes 3 months of free Level I support.
(2) Includes UNIX System V & Documenter's Workbench. Available at no cost to only those customers who upgrade to UNIX System V, Release 2.0.

Sub Licensing Fee Schedules

	UNIX System V	WWB	DWB	MC6800 SGS	BASIC	"S" STATISTICAL
Up-Front Fee	$25,000	$3,000	$3,000	$10,000	$10,000	$5,000
Binary Sublicensing Fees						
1-2 User System	60	50	10	50	50	100
1-8 User System	125	100	15	100	100	200
1-16 User System	500	150	30	250	250	400
1-32 User System	1000	250	45	500	500	800
1-64 User System	3500	350	125	1500	1000	1400
64+ User System	7000	550	250	2500	1500	2000

Discounts of 2% for each $100,000 of sales commitment may be taken in sublicensing fees up to a maximum discount of $3 million or 60%.

Teletype 5620 Terminals

Item	Commercial	Educational
Core Package Source	$5,000.00	$5,000.00
Core Package Binary Code	$1,600.00	$1,600.00
Core Package Right-to-Copy	$800.00	$800.00
Development Package Source	$15,000.00	$15,000.00
Development Package Binary Core	$6,000.00	$6,000.00
Development Package Right-to-Copy	$3,000.00	$3,000.00
Text Processing Package Source	$3,000.00	$3,000.00
Text Processing Package Binary Code	$1,000.00	$1,000.00
Text Processing Package Right-to-Copy	$500.00	$500.00

C Language Compilers

	First CPU	Additional CPU's	Sublicensing Fees
C/370	$4000.00	$2000.00	$200.00
C/SEL	$4000.00	$2000.00	$200.00
C/6000	$4000.00	$2000.00	$200.00
C/Data	$5000.00	$2500.00	$200.00
C/Cray	$5000.00	$2500.00	$200.00
Fortran 77	$3000.00	$1500.00	$200.00

Other Commercial

	First CPU	Additional CPU's	Sublicensing Fees
Photo typesetter Programmers Workbench	$3000.00	$1500.00	$200.00
Photo Typesetter V7	$3300.00	$1100.00	$200.00
Independent Troff	$4000.00	$2000.00	$200.00

Other Educational

	First CPU	Additional CPU's	Sublicensing Fees
Photo Typesetter Programmers Workbench	$400.00	$200.00	-
Photo Typesetter V7	$400.00	$200.00	-
Independent Troff	$400.00	$200.00	-

Other Educational - Administrative

	First CPU	Additional CPU's	Sublicensing Fees
Photo Typesetter Programmers Workbench	$2000.00	$700.00	-
Photo Typesetter V7	$2000.00	$700.00	-
Independent Troff	$2000.00	$700.00	-

The Future of UNIX

John Lions
University of New South Wales

[A talk to the Australian UNIX Users' Group,
Sydney, February 20, 1984.]

I have been asked to speak on the future of the UNIX system. This will be a personal view of course, and in order that I may not immediately be proved wrong by the revelation of commercial decisions already taken elsewhere, I want to talk about the UNIX system as it might be ten years from now. Predicting ten years ahead seems much easier than predicting only ten months - ten years is beyond the planning horizons of all but the largest or most confident companies in this business. And it is possible to get ten year plans very, very wrong, as anyone in this state's Electricity Commission today can tell you. No doubt companies such as IBM attempt to plan ten years ahead - but I would be prepared to wager a large amount of money that their ten year plan for 1974 made no mention of the UNIX system.

1974 - ten years ago - was the year that UNIX went international. The July, 1974 issue of the Communications of the ACM carried to the world at large the paper by Dennis Ritchie & Ken Thompson entitled ``The UNIX Timesharing System''. This had originally been delivered on October 16, 1973, at the Fourth ACM Symposium on Operating System Principles held at the IBM Thomas J. Watson Research Center, at Yorktown Heights, New York. So IBM certainly had heard of UNIX by 1974.

Ritchie & Thompson's paper has been revised and reprinted on several occasions. Whereas the 1974 version begins: ``There have been three versions of UNIX.'', the 1978 version states: ``There have been four versions of the UNIX time-sharing system.'' Note the subtle change in wording: not only had a new portable version of UNIX been developed that then ran on the Interdata 8/32, but sometime around 1976, the word UNIX had become aggrandised to become a trademark of Bell Laboratories. As the lawyers will tell you, a trademark is an adjective, not a noun, that must be applied to something that can be made and sold. So will there still be UNIX programmers ten years from now ?

In 1974, the announcement of yet another operating system was not overly exciting, whereas ten years earlier, introduction of a new operating system was a major media event. I would venture to predict that again by 1994, any announcement by a credible authority of a major new operating system will once again be newsworthy, because it will be so rare an event.

The 1974 paper was important to us at the University of New South Wales because it coincided with the decision by the university to replace its existing computer by a CDC Cyber mainframe and brace of DEC's PDP11/40 computers. Serendipity. UNIX became available to us just when we were about to take some bold new steps, and before our user community had developed new addictions to some of the more mediocre software alternatives that, like the poor, seem always to be with us. UNIX was important to us because it allowed us to exploit our own facilities, especially in Electrical Engineering, more efficiently. It also allowed us to deploy our available manpower much more effectively (but that is another story ... I digress).

If we are going to look a long way forwards, then we should first prepare by looking a long way back. History is not bunk. If we do not look back then we will not judge correctly the directions for change, nor accurately assess their force. This need was reinforced strongly for me recently: I had asked groups of third year students in Computer Science to prepare proposals for a computer network development that might serve our university until the year 1990. I was dismayed to find that they projected that the number of computer terminals on campus would only double or triple in that period. Whereas in the last decade, the number of terminals has gone from about 20 to about 500 - an increase by a factor of 25. Perhaps the students may be right in one sense - the number of terminals for student use may not grow by more than three times - but this will only be because the students themselves will have their own personal computers, a possibility that most of last year's students overlooked entirely.

The Past.

How far should one go back in time? Forty years is too far: in 1944 Colossus was working for the British against the Germans, but nobody knew. In 1944 in Philadelphia, the ENIAC was being constructed with all of 20 ten-digit storage registers. And the EDVAC, which was to have 2000 words of memory, had been proposed.

Ten years later, in 1954, much had changed, but computers were still modest in their capabilities, and were still very much the exclusive territory of those computing aboriginals, the numerical analysts. The economics of computer programming was already a matter for concern: one of the first automatic algebraic programming systems was introduced by Laning and Zierler for the MIT Whirlwind computer, which had 1024 16 bit words of memory. Backus and company had started work on the first Fortran compiler for the IBM 704. The cost of memory systems still dominated both programming and computer architectural design. Prospects for operating systems - good or bad - hardly existed.

It didn't take too long for things to improve however, and by 1959, the idea of timesharing was suggested formally as a way of improving the productivity of computer programmers, for the first time by Christopher Strachey.

A further five year jump forward in time brings us to 1964, just 20 years ago: several important events happened that year but one event was of undoubted importance to nearly everybody: the announcement by IBM on April 7, of its System 360 line of computers. (We now know that Edsger Dijkstra regards this as one of the blackest days in the history of computing.) But, for better or worse, in spite of gross defects in architectural design that have now been untidily plastered over, the design continues to this day as the basis of a commercially successful line of computers that has been widely imitated. (We also know that Dijkstra regards the copying of the /360 design by the Russians as one of the greatest coups of the cold war.)

There are several aspects of the IBM/360 that are worthy of comment here:

1. for the first time, a single computer architecture was being presented for a set of computers whose raw power varied by a factor of fifty to one.

2. by using main memory inefficiently it created an insatiable market for memory devices, and a guaranteed source of income for memory manufacturers. This market has remained incredibly active and competitive until today, with benefits for all, and will remain so for the foreseeable future.

3. it sanctified the eight bit character, distinguished it forever by the name `byte'. It legitimised the concept of character addressable memory as an alternative to word addressable memory that had been in vogue since the time of Charles Babbage.

4. it introduced the moving head disk drive as a commercial reality, and with it, the era of low cost random access mass storage.

5. it legitimised the concept of a multiprogramming operating system.

For their sins, IBM did not deliver multiprogramming support until 1967, then only in the form of a totally gauche system called MFT II. But by announcing it in 1964, they saved the bacon of several manufacturers such as Burroughs who were encountering sales resistance in selling their already working multi-programming systems to sceptical customers. Once again, in 1984, IBM seems to have found the need to come to the aid of the party ... but does A.T.&T. really need their help ?

In their 1964 announcement IBM overlooked the attractions and usefulness of both virtual memory (as it came to be called) and time-sharing. They thus missed participating in a major development: the world's most ambitious timesharing system, Multics, a joint project involving MIT's Project MAC, the General Electric Company, and Bell Laboratories.

If we look forward another five years, to 1969, now ten years after Strachey, we find that, while Bell Laboratories had withdrawn officially from the Multics project, a small group of researchers at Murray Hill were burning the midnight oil, trying to build a modest replacement - which was to become the system that we now know as UNIX.

But before leaving the /360, there is a personal postscript: in 1966, IBM shipped Gene Amdahl, chief architect of the /360, around North America on a good-will fence-mending tour (they figured that if they couldn't ship the software, then perhaps some liveware would keep the troops calm). He said many things but one statement that I remember particularly, because I strongly disagreed at the time, was that the way to build a powerful system was to use only one CPU, and to invest all your efforts into making this as fast as possible.

Back once more to 1974: this was the time of proprietary operating systems - when each manufacturer sought to convince the marketplace that not only was its hardware better in every way than that of its competitors, but that its operating system, and also its Fortran compiler, was the best there was. UNIX, in the role of yet another orphan operating system, and almost naked without its Fortran compiler, had a hard road to hoe. The recent history of UNIX does not bear repetition in detail now: it survived and prospered because it did a lot of things right, and it did hardly any things wrong, and it did some things not at all.

The Present

UNIX has often been criticised because of features that it does not have. In most cases, where a prized feature of some other system is missing (take index sequential files as an example), one can be reasonably sure that:

1. Dennis and Ken did consider it as a possibility;

2. either they decided that it wasn't such a good idea after all (consistency and economy were important goals); or

3. they have not been satisfied by any of the proposed ways for implementing the concept (e.g. inter-process message passing).

In 1979, five years ago, the Seventh Edition of UNIX was released by Bell Laboratories. This was the last release to be prepared for external distribution under the supervision of Ken Thompson and Dennis Ritchie in the Computing Science Research Center of Bell Labs. Since then major organisations have been developed both within Bell Labs and also at UC Berkeley to do the same task. In 1979, Dennis Ritchie came to Australia and explained how many of the things in UNIX had developed the way they have. In that year also, I went into print saying that I believed ``the UNIX system is a phenomenon whose full influence has not yet been experienced''. I certainly do not feel any need to retract that statement now.

Today, in 1984, there exists an ecological niche for a standard, machine independent style of operating system that can be applied to a wide range of machine sizes. This niche has existed for some time, but few perceived it. It has always been there just as long as there have been talented engineers around with new ideas on how to build better and cheaper computers. CDC, SDS, DEC, Interdata, Prime all started that way - and to a greater and less extent, because there was no other way, they have all faced and then stumbled over the software hurdle. New entrants in today's race, such as Pyramid and ELXSI, don't have to jump the same hurdle, provided they can push past all the other competitors trying leave via the gate marked UNIX. Why should UNIX be the one to fill this niche ? Because it is the first system to get its internal structures approximately correct, but nevertheless to conceal these effectively so that they can evolve. Also it has gained wide-spread acceptance just at the time that the need to fill the niche has become really pressing. Cynics might say that there is another reason: that during its formative years, it totally lacked the quote-unquote support of any major computer manufacturer. Imagine what might have happened if DEC had decided to jump on the UNIX bandwagon in the mid-1970's. They might have been able to smother it with kindness!

It is already trite to suggest that UNIX will become a de facto standard - the domino effect has already started. Soon, once we have outgrown the present generation of personal computers, which are still a little too modest and toylike in their capacity, UNIX will outpace CP/M and MS/DOS and such like, and become the de facto standard for personal computers. Since Cray computers have already announced their intention to join the fold, we can now see one software architecture spanning a range of computer systems, whose powers vary by a factor at least one thousand - a significant achievement by any standards, and one that IBM would gladly claim, if it were its own.

The Future

The future of UNIX as a widely used system is now secure, and will remain so for several years. There are several questions we may ask:

1. will UNIX survive as long as ten years ?

2. if so, will it be subject to serious challenge by 1994 ?

3. if so, where will this challenge come from ? will it come from without or within ?

4. how will UNIX change in the next ten years ? will it survive intact or only in parts ?

5. will it be able to adapt fast enough ?

6. will it be able to cleanse itself of accretions contributed by legions of well-meaning system hackers ?

I am not sure in what order I should attempt to answer these questions:

Technology.

Throughout the history of computing, the dominant force for change has been the evolution of hardware technology, mostly in the USA. VLSI, the latest dominant technology is almost unique in that, if you want to make something faster or more reliable or more efficient or more economical, then you should make it smaller ! Everything points in the same direction. There are not the tradeoffs of speed v. efficiency, or safety v. reliability, say, that confront the automobile designer. This fortuitous circumstance has created opportunities for innovators to cash in, leading to phenomena like Silicon Valley - the cancer of modern capitalism - and the rapid developments that we now regard, not as surprising, but as inevitable and expected.

Microchips are already incredibly cheap - take a walk around your local hardware store and find how few low technology items you can buy today for less than $10. But, as hardware technology innovation is not slowing, the next decade should bring yet another hundredfold or more improvement in CPU and related device price/performance. I am reminded of the story about the customer in the Rolls Royce dealership who asked the salesman how many horsepower did the engine have?. And in measured tones, the salesman replied ``Enough sir, enough´´. While most of us will have to stick to our Mazdas or Toyotas on the roads, I see no reason why, by 1994, we should not all have the equivalent of a Rolls Royce personal work station.

By 1994, even modest personal computers will have a few megabytes of main storage, and several MIPS of processor power. New ways of soaking up these resources will have to be found: program swapping to and from secondary storage, once the hallmark of all timesharing systems, has already almost passed into history. The UNIX system block buffer cache has long ago expanded way beyond the modest recommended 15 blocks, or rule of thumb ´two per user´, for Sixth Edition UNIX. But there must be a cache size (say one megabyte on a personal computer) beyond which it is not worth going. Large bitmaps for controlling displays on CRT screens will consume much storage in future, but one megabyte per screen should be ample for most purposes. That probably still leaves a few megabytes to go, so there will still be a problem. If it

is going to adapt to this, UNIX will have to learn not to throw away information, just because no one seems to be using it anymore. For example, program texts (even without the sticky bit) will stay around in main memory even when no one is executing them. Thus there will have to be changes in the way UNIX assigns and manages resources - it will have to learn to be much more laid back - but there is no real difficulty here !

Organisation.

Changes in hardware technology have always been accompanied by changes in computer architecture and organisation. But while technology has contributed improvements in price performance of the order of 10 to power 8 over the last 40 years, improved organisation has contributed, charitably, at most 10 squared. Register sets have been expanded, replicated, generalised, specialised, hidden and revealed; instruction sets have been enriched and expanded (but now it is fashionable to contract them again); pipeline techniques have been used to reduce effective instruction execution times; and memory accessing and addressing schemes have been greatly improved.

But today, in 1984, things seem ready to change: economies of scale in manufacture favour the small CPU more than the large one, and Grosch's law has been repealed. New approaches to computer organisation have become feasible and attractive. There seem to be many people, with virtual shoeboxes full of microprocessors, wondering what would be the best way to string these together to create super-fast, high capacity calculators. But as the designers of Illiac IV will verify, it is much easier to propose than to deliver. Such systems pose new problems for the operating system designer. The problem of course depends on the style of organisation chosen.

For so-called tightly coupled systems, where several processors share access to a common memory, significant changes (equivalent to the addition of sets of Dijkstra's P & V operations) need to be made to the UNIX kernel. These can be done; and have been done effectively. Thus there is no theoretical difficulty in using UNIX in the traditional multiprocessor configuration.

But multiport memory is expensive, and not suitable for more than a handful of processors at a time. There is an alternative system model that is frequently proposed: a set of processors each with its own private memory, but able to communicate with the others via a suitable network e.g. an Ethernet channel. This model is easy to describe and seems to be conceptually tidy, especially at the hardware level. However it suffers greatly from the problems inherent in using messages for communication between sequential processes. There are so many things that can go wrong when messages are passed around among groups of potentially unreliable devices that the associated protocols are heavy handed, and such systems have, in my limited experience proved to be constipated, and a disappointment to their designers and advocates. Many people seem determined to build and use such systems. Surprisingly, many of these are using UNIX as the basis of their software development even though, as has often been observed, UNIX is lacking in inter-process communication facilities. My attitude here is best summed up by saying that I have become converted to Amdahl's viewpoint: no matter how attractive the multiprocessor solution appears performancewise, a single processor solution will be better.

Networks.

But this is not to say that networks will all wither and fade away. Far from it. UNIX itself has already spawned a new phenomenon - the loosely-ravelled, lightly coupled network exemplified in Australia by the SUN network, and in North America by USENET. The distinguishing features of such networks are:

1. limited bandwidth connections;

2. mixed processor and software types;

3. owned and controlled by groups with vastly different management styles and viewpoints;

4. cooperation is for data transfer, and not in general, for load sharing.

Networks such as the SUN allow remote logins and hence a limited amount of load sharing. Personally, I find this only of interest, and hence start to migrate around the network, when the system I happen to be using has become depressingly slow (rare) or because it has ceased to function. (This was true when I was preparing this, which happened to coincide with an overdose of preventive maintenance for the machines in our laboratory. I never did locate a peaceful machine that day. But before the day was finished, I had generated an explosion of copies of all the files of current interest to all four machines that I use. And then, with the pain still fresh in my memory, I did the same again the next day. Now, sometime soon, before I forget, I will have to go around and tidy up the pieces.)

File Sharing.

While I seriously believe that load-sharing will not be a major consideration in the networks of the future - most of us don't really know how to keep the equivalent of three or four VAXes busy - I do believe that file sharing will be of prime importance. Attempts have already been made to unify the naming of files across groups of systems, with varying degrees of success. But I ask you to consider whether, given that the networks of the future will consist not of dozens but thousands of machines, whether a simple, apparently hierarchical scheme will be sufficient. As an example, suppose someone has removed my copy of the program `adventure', but I know that somewhere out there, someone else is sure to have a copy. Should I be allowed to execute (as presumably I may under the Newcastle Connection) the command

```
cp  /../*/bin/adventure   /bin
```

and hope for a reasonable result ?

As long as communication channels are less than perfect and of less than infinite capacity, then people are going to want to make local, accessible copies of distant data objects. We will want to make, and keep temporarily, local copies of distant files. Eventually, getting rid of these files - the garbage collection problem - will become serious. There are programs, for example `uudiff' that can determine whether two apparently similar files on different machines are identical. When such a pair is detected, then we can most probably happily abandon one of the copies. But what if they are only almost identical - which one should we keep ? or should we keep both ? who

decides ?

There is another technology-derived dimension to this whole problem. In a few years, information will no doubt be stored more economically on-line than as black marks on white paper. Ten years from now, it will undoubtedly be possible economically for any person to keep on-line, if he or she wishes, a file copy of every major document or program he or she ever writes. I suggest that if you attempt to do this, then you will have a real database problem on your front door-step. Does anyone sell real personal database systems yet ?

Perhaps you regard problems such as these as minor. I suggest that ten years from now they will have reached plague proportions. The solution may have to be drastic: a complete reevaluation of files, and how to manage them. If you reread Dennis Ritchie's 1979 paper, you will find that, in a real sense, that is where Ken Thompson and Dennis, with some help from Rudd Canaday, all came in - fifteen years ago.

Bytes or Bits

There is another aspect of the UNIX system to which I wish to draw your attention: the internal structure of UNIX files. As you know, there is almost none - a UNIX file is just a sequence of characters. Officially, any character set will do, but if you look more closely, it is fairly clear in the present implementation in several places that these had better be eight bit ASCII characters. UNIX programmers are only half joking when they refer to their CRT terminals as glass teletypes. UNIX may have thrown over the yoke of the Hollerith card - we should all be grateful that 80 is not a magic constant anywhere in UNIX - but it is tied to at least one aspect of mid-century electromechanical hardware. So far this has not been irksome, but I think it soon will become so once the new generation of bit-mapped terminals really arrives. The latest version of `troff' represents characters internally as 32 bit quantities, because there is size and font information, as well as a value, to be accommodated. In the future we may also want to represent colour and texture, for example. What will be the best character size then ?

It seems to me the only dependable magic constant is the value one. A machine with a bit-addressable memory is going to be attractive both for dealing with arbitrary communication links and for managing bit-mapped graphic displays. Sooner or later, some silicon packer is going to do the obvious thing and produce a microprocessor with 32 bit addresses, and addressing down to the individual bit. The Burroughs 1700 system showed how it could be done, and if Burroughs sales and marketing had not been so completely inept, such systems might have been commonplace already.

If machines that deal with variable length bit strings as their stock in trade do become common - and I believe there are strong reasons to investigate them further - then this could be one development that will rock the present UNIX design to its foundations. I am sure of one thing: in the new born-again UNIX file system, files will achieve their ultimate evolution as just `a sequence of bits'.

The User Interface

In the coming decade, some features of the UNIX system are going to survive better than others: some will emerge almost unscathed; others will

have become almost unrecognisable. The user interface is one of these.

Many new users of the UNIX system are unhappy with the user interface. A reasonable response from an old hand to a new chum making such a complaint is to suggest that he might like to change it (after all the user interface is not set in concrete). By the time the user is capable of remodelling the interface he usually doesn't worry anymore. I am told that this is not what is usually meant by a user friendly interface !

The Bourne shell represents a landmark in control language design: it should not be criticised because, for instance, it does not maintain and edit scripts of commands entered previously. Such features are useful, desirable, and should exist. But they do not belong in the shell program, because they are more generally useful than just that. They belong in yet another layer (should we call it the veneer ?) of the user interface.

Such arguments aside, the standard UNIX interface is soon to be transformed by the arrival of the new generation of terminals with bit-mapped displays. For reasons that can't be discussed here, the so-called Blit terminals have not yet escaped outside Bell Labs, but within the Labs they are already proving a potent force for change. Keep watching for further developments. The standard UNIX user interface is not going to be what it used to be.

What else?

There are many other aspects of the UNIX system that can be singled out for individual comment besides the file system or the user command interpreter and its accompanying conventions.

For many people, UNIX is the set of commands in Section One of the UNIX Programmer's Manual. Many of these, thanks to the efforts of Brian Kernighan and Bill Plauger, have already been born again as ``software tools''. Commands such as grep, awk and diff are based on the results of serious original research, and represent important contributions to computing technology. I would venture a bet that by 1994, the Oxford English Dictionary will include an entry for `grep': a synonym for search sequentially for a particular pattern.

The number of programs that are candidates for inclusion in Section One may be expected to grow enormously - one or two decent Fortran compilers may even be among them.

For some users, the important parts of UNIX are defined in Section Two of the Programmer's Manual, not in Section One. This defines the interface between user programs and the kernel, and represents the place where hackers like to hack. Standardisation of UNIX means above all, standardisation of this interface. There is a real chance that the current standardisation effort may succeed. There are both good and bad aspects to this.

Finally, there is the matter of the UNIX philosophy - will it survive ? Many newcomers to the scene hardly seem to know that it even exists. Perhaps, like the chivalry of the knights and heralds of centuries past, it will be quietly assigned to the dusty pages of history. So let me remind you that there is a UNIX philosophy, and that it is precious, subtle, tantalising, and fragile.

I was appalled recently when the representative of one computer manufacturer, newly jumping on the UNIX bandwagon, announced that ``of course they would be rewriting all the entries in the programmer's manual to conform to the company's corporate standards.'' I always thought that the erudition, charm, humour and conciseness of the UPM was one of UNIX's secret weapons: a real breath of fresh air compared to the competition.

One of the greatest threats to the original UNIX tradition is what I will, unfairly no doubt, refer to as the Berkeley UNIX tradition. Rob Pike tried to demonstrate this in his recent talk to UNIFORUM entitled `Cat -v considered harmful'. The original UNIX tradition is not to overload commands with armsfull of options, so that each can do a multitude of tasks badly, but to find a modest set of tools that can be usefully combined to do many tasks well. That is why history files etc. do not belong as part of the shell - they are potentially much more useful if they can be provided as a separate facility that may be used elsewhere as well.

A similar argument may be used to suggest that record locking should not be part of the UNIX file system. Since UNIX files can reasonably be as small as one likes, they can be as small as anything worth locking individually. And if you can then show that directory searches will really become too expensive, then there are certainly ways to speed up directory searches. The important thing is to identify the real problem.

Likewise many simple database enquiries can already be solved satisfactorily by a straightforward application of `grep' to a sequential file. But remember, it is the application that is straightforward, not the `grep' program itself.

In summary, UNIX will change in the next ten years, both for better and for worse. In its new-found maturity, it will surely miss the careful nurturing of Dennis and Ken: perhaps their greatest contribution has been the many attractive, but inessential features that they left out of their design. It seems inevitable that with so many well-meaning but undisciplined people now out to add their own improvements, the average quality of the system will decline, but not too disastrously, I hope.

Hardware and other developments are possible in the next ten years that will invalidate, or challenge, many of the basic assumptions on which most systems rest today. Challenges from new system designs are also inevitable so long as there are people with bright ideas and enthusiasm. It is rumoured that Ken Thompson is once again thinking about a new system design: if he does produce it, then I am sure we should all sit up and take notice.

Implementing The F77 Compiler on a PDP-11/34*

Roy R. Rankin
School of Electrical Engineering
University of Sydney

1. Introduction

F77 is a FORTRAN compiler from Bell Laboratories which generates code for the second pass of either the Ritchie or portable C compilers. Thus the code generated from f77 is compatible with code generated from the C compiler and the C libraries libc and libm are used as well as separate FORTRAN libraries.

The problem with running f77 on a PDP-11/34 or other processors with 64Kb address space instruction space only is lack of memory. The size of the first pass of the f77 compiler(f77pass1) is 60Kb of text and 30Kb of data. In addition, more data space is required for dynamically allocated tables. In order to run the program, the amount of data must be reduced and the text overlayed.

2. Overlaying

F77pass1 was overlayed using the Haley, Joy and Jolitz overlay system.1 In this system, each procedure has a "thunk" in the base region of the overlayed program. Calls to the procedure go to its "thunk" rather than its entry point. The "thunk" determines if the procedure is in the currently mapped segment or the base region. If it is not, it makes a system call which maps in the proper segment by changing the segmentation registers. The entry point of the routine is then called. The procedure then calls a version of csv which puts the previous segment number onto the stack. As this alters the size of the stack frame, a separate set of libraries must be compiled which pickup their arguments from the proper locations. In addition, as many system calls which are written in assembler do not call csv, these must be located in the base region. In this overlay system, up to 7 overlays are allowed.

To implement the overlay system, changes must be made to the UNIX** kernel, cc, c0, c1, ld, and as. Except for as, the overlay feature is invoked by flags so that the overlay version of the other programs can be used for non-overlayed programs. System utilities adb, file, nm, size, and strip must be modified to handle overlayed programs. Because overlayed programs have their own magic numbers, overlay programs can be recognized by the utilities so the utilities can be used on both overlayed and non-overlayed programs. As noted above, the C libraries must be recompiled in an overlay version to account for the change in the size of the stack frame.

On PDP-11's with instruction space only, there are 8 segmentation registers each of which can map up to 8Kb of address space. These

---

* PDP is a trademark of Digital Equipment Corporation

1. Haley, Charles, William Joy, and William Jolitz, "Running Large Text Processes on Small UNIX Systems".

** UNIX is a trademark of Bell Laboratories.

segmentation registers must be divided into 4 regions, base, overlay, stack, and data, and only the segmentation registers in the overlay region is altered during execution. In overlaying f77pass1, one segmentation register was used for the base region, one for the stack region, and two were used for the overlay region. This leaves 4 segmentation registers for data. Thus we have available 120Kb of text space (1 base register and 2*7 overlay registers) and 32Kb of data space.

In order to fit the compiler into the allowable space, three tricks have to be performed. The first trick is to move data which is not changed during f77pass1 execution into the text region. Yacc, which is used to generate the compiler, generates large amounts of such data. The code from yacc is compiled to assembler where it is edited to move selected tables of data to text. The second trick is to remove the text of the error messages from the compiler to a disk file. The program mkstr does this operation and replaces the error string by a pointer to the error message location in a disk file. The third trick is required to keep the base region to 1 segmentation register. As noted above, the system call routines must be located into the base region. If all the library routines are put in the base region, however, the size of the base region exceeds 8Kb. The solution was to make a separate stdio library which could be loaded into an overlay segment. Thus the base could contain the "thunks" and the remainder of the library routines without being larger than 8Kb. With these tricks, a working compiler was generated.

3. Benchmarks

The following benchmark was used to compare the f77 compiler against our current FORTRAN-4 compiler which is called fortran.

```
c       Fortran benchmark
        p=3.141592654
        a=0
        n=10000
        write(6,1)
1       format('Start benchmark')
        do 20 i=1,n
        f = p/n
        x = f*float(i)
        e=abs(alog(exp(x))/x)-sqrt(sin(x)**2+cos(x)*cos(x))
        a=a+abs(e)
20      continue
        write(6,2)
2       format('End benchmark')
        end
```

Table 1 shows the performance of the f77 compiler as compared to the fortran compiler.

Compiler Benchmark

Compiler	fortran	f77
Compile & link	3.8u 5.1s	6.6u 13.6s
Execute	23.9u 0.3s	52.1u 0.5s
Size	6,742	26,784

TABLE 1. Comparison of fortran and f77 compilers

As can be seen the f77 takes twice as long to compile and link, twice as long to execute and is 20,000 bytes bigger. The reason for the slow execution and large size was then investigated.

4. Performance Enhancements

Mosher and Corbett2 compared the performance of F77 to VMS* FORTRAN on a VAX* and made a number improvements in speed. As a version of f77 which incorporated these improvements was not available, and some of the improvements were specific to the VAX, an investigation was made of improvements for the PDP-11. It was discovered that f77(and C) are very inefficient on single precision floating point computations. Table 2 shows the code generated for a simple floating point expression with the times required to execute the code.

a = a + a*b

float	double
movof -12(r5),r0	movf -16(r5),r0
movof -16(r5),f1	mulf -26(r5),r0
mulf r1,r0	addf -16(r5),r0
movof -12(r5),r1	movf r0,-16(r5)
addf r1,r0	
movfo r0,-12(r5)	
60 microsec	52.5 microsec

TABLE 2. Floating point code generated by Ritchie C compiler

In both cases the floating point processor is in double precision mode. As can be seen the single precision version requires more bytes of code and executes more slowly than the double precision version. As producing good single precision code would require major changes to f77pass1 and c1, this problem was not addressed further.

For the basic math functions alog,dlog exp,dexp sin,dsin cos,dcos sqrt,dsqrt and pow_rr,pow_dd the assembler routines were taken from the

2. Mosher, David A. and Robert P. Corbett, "F77 Performance", Vol IV No IV, AUUGN, July 1982.

* VMS and VAX are trademarks of Digital Equipment Corporation

fortran library and the argument passing was changed to conform with f77. For the single precision routines the mode of the floating point processor was saved and then set to single precision, and restored on exit from the routine. This allows maximum execution speed in the single precision routines. Altering the compiler to call the appropriate routine was very easy. It was found that including single and double precision routines greatly increased the execution speed of the benchmark.

A second significant increase in speed was achieved by changing the default integer size from 32 to 16 bits. This was found to significantly reduce the overhead in both memory and time on DO loops. Although the change in integer size contravenes the f77 standard, our present fortran compiler does not even support 32 bit integers and the default integer size for f77 is switch selectable at compile time, so that the advantages seem to outweigh the disadvantages.

Internal to the f77 compiler and f77 libraries there is a data type ftnint which was declared as long. There is no good reason for it being type long, and if it is declared as type int a reduction of about 3000 bytes in the size of the library can be achieved. The library is still much too big, but a major rewrite would be required to reduce its size.

With the above changes made, the benchmark was run again. The results of this is shown in table 3.

Present Performance

Compiler	fortran	f77
Benchmark		
Time	23.9u 0.3s	22.4u 0.5s
Size	6,742	23,976
Onedim		
Time	144.7u 0.8s	135.3u 1.0s
Size	34,468	55,692

TABLE 3. Performance with new math routines and 16 bit integers.

As can be seen the benchmark now executes at a speed slightly better than the version compiled by the fortran compiler, but too much is still given away in program size. Also in Table 3 an engineering production FORTRAN program called onedim was run. Execution time for the f77 compiled version is again seen to be slightly better than the fortran compiled version.

5. Conclusions

1. F77 can be run on PDP-11's with instruction space only using an overlay loader.

2. Rewriting math functions and using 16 bit integers provided a doubling in execution speed for a floating point benchmark.

3. Compiling with f77 is slow.

4. Programs 20K bytes too large due to f77 libraries which need a total rewrite.

5. F77 (and C) need to handle single precision floating calculations better.

Interfacing the Quadritek 1600 Photo-typesetter to Troff

Glynn W. Peady
Australian Atomic Energy Commission
Lucas Heights Research Laboratories

Summary of presentation to the Australian UNIX Users' Group
Summer Meeting - 1984

The Quadritek 1600 Photo-typesetter is a stand-alone photo-typesetting system consisting of an entry station (keyboard, screen and disk storage) and a photo-typesetting unit. The system is similar to the CAT photo-typesetter for which Troff was originally designed. It has a mechanical system for changing point sizes and photographically recorded fonts. Of course it is not the same as the CAT which appears to be no longer available.

The suppliers of the system were prepared to deliver the system on the understanding that if it was demonstrated that the system could not be interfaced to Troff then it could be returned at no cost. This offer was too good to refuse and the challenge was taken up.

There were two possible means for interfacing to Troff: to make the Troff output appear as though it had been entered by the typesetting system entry station, or to directly interface to the typesetter unit. Since, initially, insufficient information was available on the typesetter unit it was agreed that the initial attempt would be to convert Troff output to a form suitable for the entry station.

The flow of information from the UNIX system to the photo-typesetter was thus:

PDP11/34 -> Entry Unit -> Floppy Disks -> Photo-typesetter
troff

This route is quite long and is the current method of operation of the photo-typesetter.

Using this method changes were necessary on the UNIX side as well as at the entry station. The Quadritek has a non standard character set which was overcome by applying a translation of characters being transferred from the UNIX system to the appropriate character in the Quadritek. The Quadritek maintains a size table for each font and assumes each character occupies the appropriate space, Troff assumes the opposite. This feature was overcome by overwriting the Quadritek size tables with all zero widths.

The UNIX side of the interfacing necessitated some changes to Troff as well as a filter to translate the output data to a suitable form for the Quadritek.

The changes to Troff were necessary to accommodate the extra characters which were available on the Quadritek photo-typesetter (ie 112), to change the mappings for the special characters and to change the sizes for the default fonts.

The filter program was needed to form the data into blocks of less than 512 bytes and to keep track of the photo-typesetter and 'troff' positions. The former is necessary as the Quadritek must move to the left hand boundary at least every 512 characters. This necessitated the output to be blocked and

a movement to the left boundary forced at the end of each block. The beginning of the next block had to then move back to the last position. The position of the typesetter caused some problems as all horizontal movements for the entry unit were expressed in size relative units. That is one increment in 12 point is twice the real size of one increment in 6 point. The only correspondence between Troff units (1/432") and typesetter units is at 6 point where 1 unit is equal to 1/432". Consequently the need for the filter program to keep track of two positions.

The current implementation is capable of setting approximately 10 characters/second. To this must be added the time necessary for troffing and transferring the data from the UNIX system and placing it on disk at the photo-typesetter.

It is intended that, in the future, the typesetter will be connected as a slave since sufficient information is now available to do this. No further changes should be necessary to Troff and only the filter will need to be changed. It will also be necessary to use a switch of some form to change the typesetter from slave to stand-alone mode.

ABSTRACTS OF PAPERS

USENIX Association Toronto Conference
Summer 83

These abstracts of presentations are given in the same order as the program. Please refer to the program for times. No abstracts are available for panel discussions. In many cases, abstracts have been edited for brevity. For the record, UNIX is a trademark of Bell Laboratories.

Keynote Address

Technology-Driven Software vs. Psychology of Users:
An Irresistible Force Meets an Immovable Object

Michael Lesk

Bell Laboratories
Murray Hill, NJ 07974
(201) 582-3000

Why do programmers discuss "users" the way district attorneys talk of "perpetrators"? UNIX commands, in particular, are thought of as designed by and for sophisticated programmers ("user-hostile", some would say). Supposing you want your work to appeal outside the super-hacker market: what should you do?

Historically, computer systems tend to increasing complexity ("creeping featurism" is a common name). We have learned to add features to software faster than we can tell which features are worth having. When there is too much choice, terseness becomes cryptic, but verbosity produces manuals you can't lift. We must deal with expert users, who want great facility and little redundancy; and novice users, who want ease of learning and constant reassurance. And we must deal with frequently used commands, for which new words (such as "grep") can be coined, and rare commands, of whose very existence the user may not have heard. With time, Unix systems are acquiring all the features, and many of the disadvantages, of much larger operating systems. In the end, greater complexity will defeat experts as well as novices: witness the Defense Department's trouble with ADA.

Systems are metaphors, and their directness and vividness are important for their understandability. Newtonian mechanics, for example, which is analogous to motions of conventional thrown or batted balls, is easier to learn than quantum mechanics, which has no such direct interpretation. Stoking a fire is easier than tuning a gas engine. Similarly, "point and move" systems are easier than command oriented systems. And systems that use words with their normal meanings (e.g. in keyword retrieval) are easier than those which introduce new meanings in rare contexts.

So what can we do for (rather than to) a user who complains that he only had one question, but to get it answered the computer asked him five more? I'll discuss several commands at Bell Labs that let non-programmers retrieve information from computers. These include an on-line library catalog, a weather service, and a national news service. We've run various experiments using these programs. Our library users, for example, have been finding books in an on-line catalog with a keyword system for almost two years, and prefer it strongly to an alternate "menu" approach. Among the principles we've found useful are:

1. *Less documentation.* Trying to fit every manual section onto a page is not a statement about writing style: it is a statement about good program design. Providing a long manual means either that people won't read it or that only a few will use your program.

2. *Fast response.* Try to avoid programs that bore the user. A lot of logging software, error message software, and the like can be dropped if the program returns fast enough.

3. *Direct feedback.* If the system responds immediately to each unit of user input, and does something, people are likely to be able to get their desired result quickly. If you must enter twenty parametersand then see the result, not only is the feedback cycle longer but the optimization in 20-dimensional space is likely to be impossible. Having every facility of TSO at your fingertips is only OK if you are a centipede.

4. *No superfluous choices.* Don't ask questions the user doesn't see the need of answering. When you want a file created, you don't really want to have to say whether it goes on odd-numbered or even-numbered blocks. Ordinary users find "Output file?" equally silly. Best is not to ask questions at all.

5. *Optimize redundancy.* If every misfingering produces a different valid command, users are insecure; but if everything must be typed three times, users get tired. People will keep their own filenames reasonable, but system designers must watch out for device and program names. Following the information density of English is a good guide: e.g., make names pronounceable.

6. *Understandable models.* There's probably no way to write a program that computes a Riccati-Bessel function which ordinary people will find valuable. Stick with programs that do something users understand.

Making compact programs was easy when very few people wrote code: there wasn't the manpower to make big disasters, only little ones. But just because system builders now come by the hundreds doesn't mean each user is 100 times as smart. Don't pride yourself on your ability to write a 10,000 line program with a 100 page manual. Pride yourself on a writing a small program that does the same job.

Looking at history, I have no doubt computers will become easier to use. Successful new technology always becomes more accessible than its predecessor. A telephone is more complicated than a telegraph, but it is easier to use. A photocopy machine uses higher technology than a printing press, but requires less skill. Computers will have to go through the same transformation, and it is up to us to push them in that direction.

Many of my suggestions, of course, are traditional Unix virtues: we've always tried to be concise, small, and fast. These are virtues both for hackers and for ordinary users. And so I hope that the dichotomy implied in the title will fade, and the simple solutions for the users will also be the elegant solutions for the hackers.

Programming Tools 1

Bcc: Runtime Checking for C Programs

Samuel C. Kendall

Delft Consulting Corporation
165 West 91st Street, Suite 2A
New York, NY 10024
(212) 624-1149
decvax!genrad!wjh12!kendall

Of the runtime errors afflicting C programs, the most common involve pointers which are null or out of bounds. The author describes a new software product, a testing and bebugging tool which catches such errors and diagnoses them in detail.

Bcc is a command used like cc(1). The design objective behind bcc included simplicity of use, portability and efficiency. In a bcc'd program, every pointer is associated with a pair of bounds; the precise and complicated details of what constitutes an error are discussed, but need not concern a beginning user. Bcc is implemented using mapc, a table-driven source-to-source translation tool. (The actual command is a shell script which coordinates mapc, a normal C compiler, and bcc's large runtime support library.) This source-to-source approach has both advantages and limitations, performance-wise and otherwise.

Some examples are included which contrast bcc's runtime error messages with the usual, generally cryptic output of erroneous programs. Bcc dramatically reduces debugging time both for experienced C programmers and for naive or trainee programmers, and its use in testing can significantly improve program reliability.

On Enhancing the Presentation of C Source Code

Ronald Baecker, Paul Breslin, and Christopher Sturgess

Human Computing Resources Corportion
Toronto Ontario Canada
!decvax!hcr!hcrvax!ron

Aaron Marcus and Michael Arent
Aaron Marcus and Associates
Berkeley California USA

Since the advent of programming, the technologies of the video display terminal and the line printer have limited the presentation of computer program source code to the use of a single type font, at a single point size, with mono-spaced characters, and sometimes without even the use of upper and lower case. The technologies of high resolution bit map displays, laser printers, and computer-driven phototypesetters allow for the production of far richer representations. embodying multiple fonts, variable point sizes, proportional letter spacing, variable word spacing and line spacing, grey scale tints, rules, and arbitrary spatial arrangements of elements on a page. The availability of these capabilities allows an entirely new approach to the presentation of source text in ways that will make it more legible, more readable, more vivid. and more memorable.

The authors have begun a two year investigation of computer program visualization, with the goals of making computer programs more attractive and more quickly understandable to a wide range of readers. We seek to make well-written but poorly presented programs more legible and readable, and to make badly-written programs more obvious. We are using the programming language C as the vehicle in which all subject programs are expressed.

Our ultimate clients, in these endeavors, are the various readers of computer programs. These include program developers. program maintainers, managers of programmers, clients of programs, users of programs, and professionals from parallel or associated disciplines.

On-line Manual System for
Software Development on UNIX

Osamu Nakamura, Jun Murai

Department of Mathematics
Keio University
3-14-1 Hiyoshi
Kohoku, Yokohama 223
JAPAN

Unix is the most popular operating system used for software development. One of the most important functions of an efficient software development environment is to provide users with information available about existing software so that users can get full benefit of using it.

There is only such information about Unix resources: the Unix manuals. The Unix manuals are usually stored in a disk space and can be accessed by the *man* command. The *man* command produces manual pages formatted for hard copy devices. Nowadays Unix terminals became CRT terminals and it is strongly required to access the manuals from their terminals interactively in a sense of *help* facilities of other operating systems.

One of the answers to these requirements might be the Berkeley's *man* command which invokes the *more* command to show a formatted manual on a CRT terminal so that users can read it by a page concept and even search for some patterns. Users also can search for a manual entry with a keyword using an option of the *man* command which further helps an unaccustomed user.

Our study described in this paper is to provide more efficient and usable functions to use information described in a

manual entry. At the same time we propose more conventions to describe a manual entry than defined in *man* nroff macros so that searching and accessing of manual information can be done more efficiently.

The on-line information system we have implemented on Unix consists of a database containing information descrbied in Unix manuals, runtime library routines to provide various functions for accessing the data from users' programs, and various commands which are used to extract information more specifically than *man,* to check C programs and to maintain the database.

cdb - A C Source Level Debugger

Michael Farley, Paul Kunkel and Trevor Thompson

Mark Williams Company
1430 W. Wrightwood
Chicago, IL 60614
(312) 472-6659

This paper describes cdb, a new C source language debugger for programs compiled with the Mark Williams C compiler. cdb lets the user trace the execution of a C program at the C statement level, and inspect its data at the C expression level.

cdb displays the C source being executed in a screen window. C statements may be executed one at a time.

A second window lets the user inspect data. Arbitrary C expressions, including type conversions and function calls. are evaluated here. Variables (including structure members) and formal function parameters are accessed by name. Full C scope rules apply.

A third window displays the output of the program, and a fourth logs trace points. Both C statements and C expressions may be traced; traced expressions trap when their value changes.

The ability to call user functions provides extensibility of the debugger. Special debugging fuctions such as display routines can be included in programs when needed.

Usually, programs run at full speed under debugger control. Thus, cdb provides strong features available only in a source level debugger without imposing the substantial cost of an interpreter.

UNIX Implementation 1

Tunis: A Portable, Unix Compatible Kernel
Written in Concurrent Euclid

R.C. Holt
M.P. Mendel
S.G. Perelgut

CSRG
Sandford Fleming Bldg.
10 King's College Rd., Rm. 2001d
University of Toronto
M5S 1A4

Tunis (Toronto UNIversity System) is a portable kernel that is compatible with UNIX. The compatibility means that existing Unix programs such as the shell, the editor, and the file utilities can (and are) run unchanged under the Tunis kernel.

The internal structure and programming of the Tunis system are entirely different from those of the original system. A great deal of effort was invested in making the Tunis system modular, understandable and easily portable.

Modularity, portability and clarity were all achieved by choosing to implement Tunis in Concurrent Euclid (CE), a higher level language than the Unix implementation language, C. CE has precisely defined semantics, elegant and portable data structures, convenient information hiding (modules) and precisely defined concurrency (processes and monitors).

The Tunis internal organization consists of four major software layers, each of which is a CE module. Within each module are CE processes that handle units of concurrency such as read-ahead on physical terminals and reallocating memory among Unix user processes. The four layers are: the *user manager* (interprets traps by user processes and relays requests to lower levels of Tunis), the *file manager* (implements Unix file systems by manipulating i-nodes), the *memory manager* (uses swapping or paging to implement users' virtual memories), and the *device manager* (isolates code that drives physical devices).

The Sol Operating System

Michel Gien

Pilot Project SOL
c/o INRIA
B.P. 105
78153 - Le Chesnay Cedex - France
Tel.: (3)954 90 21
devax!mcvax!vmucnam!mg

The SOL* operating system provides a portable Unix* environment implemented in standard Pascal. It is operational on Honeywell / Level 6 mini computers and SM90, a 68000 based micro-computer developed at CNET**. It is being ported on other mini and micro-computers (in particular those based on 68000, 8086 and 16000 microprocessors).

The SOL operating system is composed of:

- a time-sharing kernel, implemented in Pascal and providing a Unix compatible interface, at the system call level.
- a number of utilities and software tools (over 150), also implemented in standard Pascal and compatible and equivalent Unix utilities.

Motivations for this development are based on the recognition of Unix as a standard Operating System for today's mini and micro-computers and Pascal as an implementation language, which level of abstraction, structured programming concepts, and standard definition, greatly enhance Unix portability, readibility and therefore maintenance and future extensibility. This development is also a step towards the association of Unix concepts with other programming language of the Pascal lineage such as Modula-2 and Ada.

The SOL operating system is not a simple C to Pascal translation of Unix. The result of such an exercise would just be a disaster. A complete re-engineering of all the internals of the system has been performed towards a more modular and adaptable structure fitting better Pascal programming facilities and constraints with the objective of improving portability and adaptability to various machine architectures without too much performance degradation.

---

* SOL is a registered trademark of Agence de l'Informatique.
** CNET : Centre National d'Etude des Telecommunications.
*** INRIA : Institut National de Recherche en Informatique
et Automatique.

An Implementation of UNIX for the Intel iAPX 286

P. L. Barrett

Senior S/W Engineer
Intel Corp. - HF2-1-800
5200 N.E. Elam Young Pkwy.
Hillsboro, OR 97123
(503) 640-7335
icalqa!omsvax!plb
Intel Corp.

This paper discusses the implementation of a UNIX kernel for the Intel iAPX 286 microprocessor. A Xenix kernel was ported to the iAPX 286 making use of the protected mode features of the microprocessor. The architectural features of iAPX 286 microprocessor used by Xenix 286 are described. In addition, the architecture of Intel systems and their relation to Xenix 286 are discussed. The overall structure of a Xenix 286 process and its relationship to the iAPX architecture is defined.

The kernel makes full use of the protection mechanism provided by the iAPX 286. System data and code are accessible only by level 0 processes and user processes run at level 3. When a process makes a kernel call, it becomes a level 0 process. A user process' address space is provided by a descriptor table that is visible only to that user. Thus, a user cannot access another user's address space without intervention by the kernel.

Xenix 286 uses a segmented memory management scheme with an allocation granularity that is a multiple of the fundamental disk block size - 1Kb. Memory fragmentation considerations are then discussed.

The Xenix 286 kernel support for multisegement programs is described. A process can exceed 64kb in text size currently and plans are underway to support more than 64kb of data.

The future directions for the Xenix 286 kernel will include support for large programs, Intel translators and higher performance I/O.

Unix a la Data General

Wayne McLaren

Data General Corporation
62 T.W. Alexander Drive
Research Triangle Park, NC 27709
(919) 549-8421

Data General has recently implemented a System III version of UNIX on the 32-bit product line (MV/4000, MV/6000, MV/8000, and MV/10000). The development of a UNIX system for products with an existing operating system and a large user base presents a broad spectrum of "challenges" to the implementors. These issues can be the result of:

* the system architecture
* UNIX documentation which can be ambiguous and/or obscure
* an attempt to achieve compatibility with existing and future proprietary products

Emphasis will be placed on the general issues and problems which will confront future developers of similar products.

Programming Tools 2

VCHK A Maintenance Program for UNIX File Hierarchies

Scott Bryan

2405 4th St.
Berkeley, CA 94710
(415)644-1230

UniSoft Corporation

UNIX systems are typically comprised of several hundred to several thousand individual files arranged in a hierarchical structure. Most of the standard programs and utilities depend on particular aspects of this structure yet are not prepared to deal with any inadequacies. Most of these dependencies are undocumented and when unsatisfied cause programs to die in mysterious ways -- usually not identifying the failed expectancy.

The *VCHK* program uses a file that describes the correct layout of part or all of the UNIX file environment. The complete directory structure, modes, ownerships, links, and information about the contents of each file may be recorded in this file. *VCHK* reads this file and makes sure the actual filesystem corresponds. Everything except corrupted data can be fixed directly by *VCHK* saving much time in the analysis and repair of mysteries due to corrupted files. The snapshot maintained by *VCHK* is a human readable file (similar to a *Makefile)* and can be altered easily to reflect new additions or alterations to the file system.

VCHK is presently used in a production environment as part of an automated system for maintaining remote UNIX facilities over phone lines.

Enhancing MAKE or Re-inventing a Rounder Wheel

Edward S. Hirgelt

Zehntel, Inc.
2625 Shadelands Drive
Walnut Creek, CA 94596
(415) 932-6900 ext. 364
decvax!sytek!zehntel!zinfandel!ed

The standard MAKE is a powerful and useful tool for software development. However, it lacks some useful facilities. In particular, there is a need to generate different versions of our products for different hardware and software configurations. Several people must modify the same software without interference to shorten the software development cycle. This paper discusses enhancements to MAKE which resolve these problems. The new features include conditionals, directory support, and special rules.

A mechanism similar to the C preprocessor's '#if' selects different pieces of the description file for interpretation. Macros may be defined by the user in response to questions issued from a description file.

The directory support mechanism decouples the source directory from the object directory. This enables us to have several object directories associated with a single source directory. Several people can then work on a subsystem simultaneously, and object files for different versions can reside in separate directories. To achieve this, the directory support mechanism maps simple file names into longer, more explicit path names.

Special rules were introduced to extend the user's control over the program construction procedure. execution of rules. A 'foreach' statement executes a block of statements once for each element in a list.

Mm4 -- Make with M4
Tools for Maintaining Makefiles

Martin J. McGowan III
William L. Anderson
Allen H. Brumm

Product Assurance Department
Computer Consoles, Inc.
97 Humboldt Street
Rochester, NY 14609
(716) 482-5000

The UNIX utility *Make* has long been recognized as an extremely useful tool for maintaining computer programs. However, for large systems, makefiles can also be quite large, and can present maintenance problems of their own. Two major problems with large makefiles are the (1) explicit declaration and maintenance of all components of targets on individual dependency lines, and (2) maintenance of many entries whose rules are instances or variations of one standard pattern.

Mm4, or make with m4, is a set of tools that solves these problems. Makefile entries for common or related tasks are captured in m4 macros. The file containing such macro invocations is called a make.m4 file. A make.m4 file is the source code for a makefile; it is processed by m4 to produce a makefile.

Using m4 macros to produce makefile entries has several benefits. First, m4 macros provide compact and readable representations of standard rules patterns. In addition, such standards are easily maintained and enforced; changes are automatically incorporated into existing makefiles. Second, the mm4 tool minc (for makes includes) analyzes C programs and nroff source files for included files, and produces a makefile dependency line for the specified target. In this way makefile dependency lines for individual targets can automatically be kept up to date. Third, another mm4 tool takes a makefile whose components are all explicitly declared, and produces a list of all primary source files and tools required to build the outputs of the makefile.

Using Make Effectively

Robert E. Novak

Pyramid Technology Corporation
2471 E. Bayshore Road
Palo Alto, CA 94303
(415) 494-2700

One reason for the popularity of Bell Laboratories' UNIX operating system is its associated utility programs. One such program, *make*, increases programmer productivity by keeping track of which programs need to be compiled or assembled to rebuild a software system. A major problem associated with the *make* facility, however, is that it requires manual maintenance of its description of the dependency information. This article describes a methodology and a utility which automatically maintains the dependency information, thereby avoiding the errors which manual maintenance introduces.

1. A Small Example of How *make* Works
2. Data Declarations vs. Data Definitions
3. Creating the Makebase
4. Creating Dependencies Automatically

gprof: A Call Graph Execution Profiler

Marshall Kirk McKusick

1616 Oxford St.
Berkeley, CA 94709
ucbvax!mckusick

Large complex programs are composed of many small routines that implement abstractions for the routines that call them. To be useful, an execution profiler must attribute execution time in a way that is significant for the logical structure of a program as well as its textual decomposition. This data must then be displayed to the user in a convenient and informative way. The gprof profiler accounts for the running time of called routines in the running time of the routines that call them. The talk describes the design and use of this profiler, which will be available on the next Berkeley distribution.

UNIX Implementation 2

File System Considerations in a Multiple Processor UNIX Environment

Ed Patriquin

Convergent Technologies
3055 Patrick Henry Drive
Santa Clara, CA 95051
(408) 980-0850

Traditionally UNIX has been run on single processor systems like the PDP 11/70. Even when UNIX moved to the world of multiple processors, it only ran on tightly coupled multiple processor systems. The MegaFrame is a loosely coupled multiple processor environment with each processor running its own copy of UNIX. These processors are connected by a high speed system bus. There are also processors in the system running CTOS, a Convergent proprietary operation system.

UNIX, in its original form, keeps a large amount of information concerning the state of the file system in in-core data structures. In a multiple processor environment this is not a feasible solution.

To provide better performance and allow sharing of file resources, the UNIX file system was offloaded and put as service process under CTOS on all File Processors which have UNIX file systems on them. This presented several major architectural problems within UNIX.

The UNIX file system was redesigned and reimplemented to run under CTOS. To make the program interface properly with CTOS, the file system was made into a message based server process. This means that the file system is now single-threaded in a multi-processor environment. A parallel server process which uses the same data structures as the regular file system but does not modify these data structures was added. Any operation which will modify a data structure is sent to the single-threaded file system server for completion. This allows the high volume traffic, such as normal reads and writes, to flow quickly around the single-threaded bottleneck. This also eliminates concurrency problems that arise with multiple threaded operation.

A High Performance Implementation of UNIX for the IBM Series/1

Michael E. Wilens

Computerized Office Services Incorporated (COSI)
313 North First Street
Ann Arbor, MI 48103
(313) 665-8778

COSI undertook the porting of UNIX System III to the IBM Series/1 as part of a joint effort with the CMI Corporation. The resulting system is called SERIX (for Series/1 UNIX) and will be announced within the next few weeks.

This talk focuses upon the capabilities of SERIX in general including several interesting aspects of UNIX on the IBM Series/1:

[1] The Series/1 is an unusual machine with a somewhat unusual architecture and instruction set. A lack of translation registers leads to unique concepts such as twilight zone cache management.

[2] A 'C' compiler capable of generating extremely efficient Series/1 code was developed. The compiler optimizes usage of hardware stacking instructions, supports inline insertion of assembly code which is subsequently optimized. and generates extremely efficient (in both time and space) object code. As a result, less than 500 lines of assembly code was written for SERIX.

[3] The I/O channel orientation of the series/1 was exploited by interleaving at both the controller and/or device levels, significantly increasing both the bandwidth (hence speed) of access to disk and the maximum size of file systems.

[4] Real-time extensions jointly developed by General Motors and COSI were added.

The Use of the Z80 I/O Processor by the TRS-XENIX Operating System

Jerry Dunietz
Robert Powell

Microsoft Corporation
The Microsoft Building
10700 Northup Way
Bellevue, WA 98004
(206) 828-8080
decvax!microsoft!jerryd

The Radio Shack Model 16 computer system uses two processors -- a Motorola Mc68000 16:32 bit processor and a Z80 8 bit processor. Only the 8 bit processor has access to I/O devices, such as the console/keyboard, built-in serial ports and parallel port, built-in floppy drive(s), and hard disk drive(s). The Z80 can access the Mc68000 memory and can generate three distinct interrupts to the Motorola processor. The TRS-XENIX system was designed to take advantage of this system architecture. The features dependent upon the dual processor design generally improve the system in one of two areas -- I/O performance and configurability.

This talk describes the design of these features in some detail. Points of interest include:

- minimizing the Mc68000 interrupt burden in high-speed serial I/O without delaying XOFF processing;
- designing one interface between the Z80 and Mc68000 to handle all character-at-a-time devices, whether serial or parallel;
- designing one interface between the Z80 and Mc68000 to handle multiple sizes and formats of hard and floppy disks;
- making the TRS-XENIX system dynamically adjust to the format of an inserted floppy disk;
- making the TRS-XENIX system auto-configure for the root and swap devices and their sizes;
- detecting the removal of a currently open floppy disk;
- for Microsoft's internal development and debugging use, providing a mechanism by which the Z80 may detect the crash of the Mc68000.

Virtual Memory Management in GENIX

Laura Neff

Microcomputer Systems Division
National Semiconductor Corporation
Santa Clara, CA
!menlo70!nsc!lneff

We have recently ported UNIX to an NS16032-based system. The port is based on 4.1bsd UNIX. Our operating system is called GENIX. It supports the virtual memory architecture provided by the NS16082 Memory Management Unit.

A GENIX process has a virtual address space of up to 16 megabytes and is mapped by a two-level hierarchy of page tables. Both process pages and second level page tables can be paged out of an faulted into memory.

The kernel's address space is mapped with its own set of page tables. Because the user and kernel page tables have the same structure, the kernel manages its own address space with the same algorithms it uses to manage user address spaces.

Sharing of texts is done by mapping executable files. When a file is executed for the first time, the kernel creates a file map for it using level 1 and 2 page tables. The process's page table entries are then set up using the file's map as a template.

User Interface 1

The Interface Arsenal:
Software Tools for User-Interface Development

Gary Perlman

Bell Labs 5D-105
600 Mountain Ave.
Murray Hill, NJ 07974
(201) 582-3624
ucbvax!mhb5b!gsp

Most programs have poorly designed user-interfaces. Most programmers do not devote much effort to developing user-interfaces. And even if they do, they have no guarantee that their work will be rewarded. UNIX programmers are not well supported when they want to develop a user-interface. The Interface Arsenal is a set of C function libraries that make it relatively painless for programmers to develop programs with user-interfaces that people find easy to use.

There are several reasons for poor user-interfaces:

- Lack of Programmer Interest
- Lack of Programmer Time
- Lack of Programmer Expertise
- Duplicated Effort (re-inventing the wheel)
- Inconsistent User-Interfaces

Having a general set of user-interface tools solves all the problems listed above. With tools that are easy for PROGRAMMERS to use, programmers save time in developing user-interfaces, and choose to use them over other alternatives such as unintelligent I/O. With software well designed from a cognitive psychological viewpoint, most problems with lack of programmer expertise are overcome. Also, programmers using a common set of user-interface function libraries do not have to spend their time duplicating the effort of others. Finally, a set of interface libraries define a standard set of user-interface conventions, which when followed, effect the development of software systems with minimal learning required by users.

The component libraries of the Interface Arsenal are:

rsvp: Runtime allocated String Variable Package
listr: List String Package
funkey: Programmable Function Keys
filer: High-Level File Handling
phmenu: Pop-up Hierarchical Menus

Future libraries will probably include general functions for type and range checking useful for form-filling.

A User Interface Management System

W. Buxton, M.R. Lamb, D. Sherman
and K.C. Smith

Rm. 2002, Sanford Fleming Bldg.
10 King's College Road
Toronto, Ontario M5S 1A4
(416) 978-6320

A User Interface Management System (UIMS) developed at the University of Toronto is presented. The system has two main components. The first is a set of tools to support the design and implementation of interactive graphics programs. The second is a run-time support package which handles interactions between the system and the user (things such as hit detection, event detection, screen updates, and procedure invocation), and provides facilities for logging user interactions for later protocol analysis.

The design/implementation tool is a preprocessor, called MENULAY, which permits the applications programmer to use interactive graphics techniques to design graphics menus and their functionality. The output of this preprocessor is high-level code which can be compiled with application-specific routines. User interactions with the resulting executable model are then handled by the run-time support package. The presentation works through an example from design to execution in a step-by-step manner.

Talking to UNIX -- Some Experience with Speech Input

Martin Tuori

DCIEM, Toronto
Box 2000
Downsview, Ontario M3M 3B9

This talk gives a brief summary of our experience to date with two voice input units, the auricle-1 from Threshold Technology Inc., and the Model 3000 from Verbex. Emphasis is on the Auricle-1, a unit for which we have developed UNIX software for training, testing and operation. The two units represent opposite ends of the voice input spectrum; the Auricle-1 recognizes only discrete words, whereas the Verbex can recognize words in continuous speech. To date we have only simple demo vocabularies for the Verbex; development software is forthcoming.

A videotape will be presented that shows each of the units in use. In the case of the Auricle-1, voice is used as input to a UNIX shell. The purpose of this introduction is to show that inexpensive voice input technology is available, and easily applied. The example of the UNIX shell indicates, however, that many common user interfaces are based on vocabularies beyond the limits of simple voice input units. The need to specify arbitrary file names makes the vocabulary unbounded, and forces us to include the spoken alphabet for voice-only input.

To alleviate this problem, we are integrating voice with keyboard and gestural input, so that each input mode may be used as appropriate. A network of cooperating concurrent processes is used, so that each input mode is continuously active. The choice of which input mode to use is left to the operator, rather than being enforced by the input grammar. An example of mixed mode input, using voice and keyboard, will conclude the videotape.

UNIX Implementation 3

A General-Purpose Object-File Format

Brian Lucas
Heinz Lycklama

INTERACTIVE Systems Corporation
1212 Seventh Street
Santa Monica, CA 90401
(213) 450-8363
decvax!yale!ima!heinz

In porting the UNIX system to various 16- and 32-bit computers and in developing cross-compilers that run on these computers but generate code for other machines, INTERACTIVE has found the object-file formats used by UNIX System III and System V, including the new *coff* format, to be inadequate. As a result, INTERACTIVE has designed a new general-purpose object-file format meant to achieve the following objectives:

(1) Support on all currently popular 16-bit and 32-bit computers.
(2) Low overhead on small machines.
(3) Common set of utilities that deal with object-file formats.
(4) Provision for compatible extensions.
(5) Independence from byte and word order.

This paper describes the details of the object-file format that achieves these objectives.

A User Information Data Base for UNIX
(What to do when /etc/passwd just isn't enough)

Clyde W. Hoover

The University of Texas Computation Center
The University of Texas at Austin
Austin, TX 78712
(512) 471-3241
eagle!ut-ngp!clyde

The increasing amount of data desired for the successful administration of UNIX systems in a large academic computing environment has outgrown the standard password file format. This need is best met by supplanting the password file with a user information data base system.

As the usage of UNIX at the University of Texas Computation Center grew, it became necessary to maintain extended accounting and administrative information for each user of our UNIX systems.

A number of methods for storing and maintaining this information were examined. The one 'unused' field in the standard UNIX password file format was found to be inadequate for this purpose; the large number of programs that depend on this format effectively disallowed changing it; the problems involved with maintaining synchronization between multiple accounting information files (/etc/passwd plus a supplemental data file), precluded that approach.

A User Information Data Base (UDB) was the best solution to this problem. This data base contains all the per-login-user information and replaces /etc/passwd for system accounting and access purposes.

The standard password file still exists (and is generated from the information in the UDB) to provide compatability with existing programs, though it is ultimately planned that the UDB will totally replace it for all applications, including user id/login name mapping (most commonly done by programs like ls).

Programs that employ the standard UNIX password file access routines can use the UDB without source changes. A library of access and manipulation routines are provided for directly using the UDB. There is also a suite of programs to keep the data base in good working order, and an interactive editor for data base maintenance and account control.

Z -- A High Performance Raster Graphics Package for UNIX Operating Systems

Steve Daniel

Microelectronics Center of North Carolina
P.O. Box 12889
Research Triangle Park, NC 27709
(919) 541 7285
decvax!mcnc!swd

Z is an outgrowth of the VLSI-CAD tools effort of the Microelectronics Center of North Carolina (MCNC). Several of the tools under development at MCNC use high performance raster-graphics processors. When their development was started no high quality, high speed, interactive, raster graphics packages were available for UNIX. A CORE graphics standard system was implemented and rejected because it was too slow and not well suited to the needs of interactive raster-graphics programs. (This CORE system, called LEO was described at the Jan 1982 Usenix conference.)

Z differs from the proposed CORE standard in several ways. First, we view the CORE system as oriented towards vector-graphics devices. CORE's concept of a picture segment maps very well onto the display lists of vector devices. The translation, scaling, and rotation of segments is easily accomplished by many vector devices. Raster devices have trouble implementing these operations.

Instead, raster devices provide such features as infinite picture complexity, the ability to erase regions of the screen, the ability to copy portions of the screen from one place to another, and the ability to manipulate the bit planes individually. Z provides access to these hardware capabilities.

Z is designed to provide support for painting on the screen, rather than creating and manipulating a graphical data base. Not only does this approach map easily onto raster-graphics hardware, but it also eliminates the time-consuming and error-prone task of mapping between the user program's graphical data base and the graphics system's data base.

Z is fast. Numerous algorithmic optimizations have been applied to make the code fast and efficient.

Attaching an Array Processor in the UNIX Environment

Clement T. Cole

Massachusetts Computer Corporation*
543 Great Rd.
Littleton, MA 01460
!genrad!masscomp!clemc

In this report, the work necessary to attach a commercial array processor, the Floating Point Systems FPS-164, to a VAX 11/780 running 4.X BSD UNIX is described. The FPS-164, its surrounding development system, and the interaction with potential application programs are also presented.

*Work done while a member of the CAD group of UC Berkeley.

User Interface 2

The Edit Shell - Combining Screen Editing with the History List

Joseph L. Steffen
Michael T. Veach

Bell Laboratories
Naperville, IL 60566
(312) 979-5381
ucbvax!ihnss!ihuxs!steffen

In an interactive terminal session you often type the same commands over and over again to edit, compile, and test your programs. Unfortunately you often make mistakes while retyping these commands so you are forced to retype the line again.

Screen editors solve the terminal input correction problem by providing commands to move the terminal cursor to any point in previous input and to delete and insert characters at this point. However, the only editing available when in the shell is the capability to erase the last character or the entire line. The Berkeley C shell (csh) provides a terminal input correction mechanism called the history list. This has editing capabilities similar to the line editor ed, but we want editing like that of a screen editor.

The edit shell gives you the desired screen editing capabilities plus access to the history list. You move back and forth through the history list with the same commands you use in a screen editor to move between lines. You can think of the edit shell as a one-line window into the history list. Two versions of the edit shell are available, one for the vi style of editing, and another for the emacs style of editing.

KSH - A Shell Programming Language

David Korn

Bell Telephone Laboratories
Murray Hill, NJ 07974
(201) 582-7975
ihnp4!mhb5b!dgk

Ksh is a Bourne shell compatible command language for the UNIX operating system. It has many added features including almost all those found in *Csh*.

The most notable additions are: a history mechanism which continues across login sessions; in-line edit capabilities, both *vi* and *emacs* style, which have access to the history; arithmetic, including one-dimensional arrays, and multiple bases; aliases and shell functions; job control on systems which support it; and a menu selection statement. Several smaller, but no less important additions have also been made.

Ksh runs faster than the Bourne shell and has many more built in commands. The shell runs on several variants of UNIX including System III, System V, BSD 4.1, and BSD 4.2.

This talk will describe these additions and the reasons for improved performance.

A Simple Window Management Facility
for the UNIX Timesharing System

David Mankins and Daniel Franklin

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, MA 02138
(617) 491-1850

A virtual terminal or "window" facility implemented on the BBN C70 minicomputer running the UNIX Version 7 operating system provides the ability to create, delete, move and change the size of (possibly) overlapping windows, and can display output in many windows simultaneously while the user is typing in another. Our goal in the design of this facility was to change the UNIX kernel as little as possible, while providing enough flexibility and power to allow implementing an efficient window system using intelligent terminals like the BBN BitGraph(1). We feel that we have met this goal. After a single person-month of work, we have a window manager running on our C/70 UNIX systems. The implementation is very efficient; when not switching between windows, it imposes no additional overhead on terminal I/O.

Our design is based on the UNIX concept of process groups, which we use to "multiplex" a single kernel terminal data structure among a user's windows. Terminal I/O is permitted only to those processes in the same process group as the terminal. A privileged user process, having nominated itself as a terminal's "window manager", handles all window manipulation.

---

(1) "BitGraph" is a trademark of the BBN Computer Company

Compilers and Languages 1

A New Portable Compiler For XENIX

Ralph Ryan, Hans Spiller, Dave Weil

10700 Northup Way
Bellevue, WA 98004
(206) 828-8080 ext. 226
decvax!microsoft!ralphr

A new portable compiler has been developed which allows compilation of multiple languages to multiple machines. This paper presents the design goals that were set up for the compiler, and discusses the success of the architecture in meeting these goals. The design of the intermediate languages is discussed including:

- support of C, Pascal, FORTRAN, and BASIC front ends.

- language and machine independence.

A case study of the first re-target is presented, including:

- problems in porting from a flat address space to a segmented architecture.

- a comparison of portability between this compiler and the portable C compiler (PCC).

Objective C

Programming Smalltalk-80 Methods in C Language

Brad J. Cox, Ph.D.

Productivity Products Incorporated
Sandy Hook, CT 06482
(203) 426-1875

The Objective-C[1] compiler provides message/object programming (a la Smalltalk-80[2]) within conventional environments such as UNIX[3]. This compiler and its library, while similar in purpose to an earlier version called OOPC[4], are a totally new design and implementation based on direct experience with Smalltalk-80.

The compiler turns Ojbective C language, a superset of standard C, into programs consistent with the run-time semantics of Smalltalk-80. The compiler makes full use of the standard C code production chain and the UNIX run-time environment. A program translator pass is added just after the C preprocessor to turn Objective C language into standard C source. It does this by performing a full parse on the incoming language, so it is capable of accurate compile-time diagnostics and can move several speed-critical operations from run-time to compile-time. The result is a language that offers the productivity benefits of Smalltalk-80 while retaining the efficiency/portability benefits of C language.

[1] *Objective-C* is a trademark of Productivity Products, Incorporated.
[2] *Smalltalk-80* is a trademark of Xerox Corporation.
[3] *UNIX* is a trademark of ATT.
[4] Brad Cox, *The Object Oriented Precompiler,* SIGPLAN Notices, January 1983

Compilers on the NS16000

Jay Zelitzky
Sunil Srivastava

National Semiconductor
Microcomputer Systems Division
135 Kern Avenue
M/S 7C-265
Sunnyvale, CA 94086
(408) 733-2600
!menlo70!nsc!myunive
!menlo70!nsc!sunil

As part of our support for the NS16000 UNIX system, we have ported the 4.1 bsd C compiler to the NS16000 and written a compiler for EPascal, a Pascal extension developed at National Semiconductor. These compilers have been designed to be used in a systems development environment. While C by nature supports systems programming, Pascal has been extended to do so. The design criteria for these compilers included the following features: good code quality, separate compilation, inter-language module linkability. These continue to be important in our plans for further development. We use the results given by some popular benchmarks to illustrate the strengths and weaknesses of the compilers and the chip set.

There are many similarities in the C and EPascal compilers' output although their internals are quite different. Both compilers generate assembly language files. Both use full 32-bit representation for integers and 64-bit IEEE standard representation for floating point numbers. The two compilers take advantage of the indexed addressing modes of NS16000 to do array indexing. The two compilers follow the same conventions for saving and restoring registers, so routines of the two languages can be linked without risk of destroying the contents of the registers.

The C compiler is an extensively modified version of the VAX C compiler. Functions returning structures are handled differently; return values are moved directly to their final destinations. The compiler efficiently supports array, pointer, and structure manipulation by using the NS16000's indexed addressing mode. It optimizes assignments to a register variable by using the register variable as a temporary register. This saves a final assignment back to the register variable since the result is already there. The C compiler uses a modified version of the VAX c2 optimizer to do some additional peephole and branching optimization.

UNIX Implementation 4

UNIX Support for Guaranteed Real-time Processing

Douglas J. Ross
ANDYNE Computing Limited
221 King Street, East
P.O. Box 1496
Kingston, Ontario
K7M 7A1
(613) 548-4355
decvax!utzoo!dciem!rds!djr

M. Martin Taylor
DCIEM
P.O. Box 2000
Downsview, Ontario
M3M 3B9
(416) 635-2048
decvax!utzoo!dciem!mmt

The PDP11 UNIX V7 kernel has been modified to support a new Interprocess Communication (IPC) mechanism and guaranteed real-time response phases for application programs. Current work is in progress to port the modifications to the Perkin-Elmer and VAX processors.

Real-time response is often regarded as "quick enough response for the job."

Often it is taken to mean that the system has rapid responses to external stimuli, and gives important processes high priority. When a set of tasks in a job are assigned different priorities, a high priority task can override a low priority one, even though the deadline for the high priority task is distant and that for the low priority task is near. Guaranteed real-time response is not achieved by a priority scheduling system, since the introduction of independently conceived tasks relating to a different job can cause a previously successful job mix to fail.

The MASCOT-TCP approach to guaranteed real-time scheduling depends on the existence of a contract between a time-critical process and the operating system. The process guarantees that it will not exceed a certain load on the system; the system combines all the loads from time-critical processes (including hardware interrupts and other sources of possible delay) and determines whether the total load can be handled. Reservations are an integral aspect of such a system, and the amount of real-time work that can be guaranteed is largely determined by the cleverness of the reservation algorithms. The current implementation incorporates a very simple and conservative reservation algorithm.

High-Speed Laboratory Data Acquisition on the MC-500

Thomas J. Teixeira

Massachusetts Computer Corporation
Littleton, MA 01460
(617) 486-9425
!decvax!genrad!masscomp!tjt

This paper describes the hardware and software architecture of the MC-500, a high-speed laboratory data acquisition system. The design goal for the system was to sample 16-bit data to disk at 1 million samples per second (limited to approximately 375k samples with currently available disk drives). This is accomplished through the use of multiple hardware buses, a specialized data acquisition processor (functioning as an intelligent DMA controller for non-DMA devices), and a modified version of the UNIX operating system running on the main processor.

The modified system supports multiple "real-time" processes that are memory resident and scheduled on an absolute priority basis, virtual memory, contiguous disk files, and a "window" system for multiple, separate graphics processors. The internal implementation of the I/O system was modified to improve file system performance and increase pipe throughput. The high-speed data acquisition is supported by multiple buffered asynchronous I/O and prioritized asynchronous system traps (AST's).

Performance Effects of Disk Subsystem Choices for VAX Systems Running 4.2BSD UNIX

Bob Kridle

Computer Systems Support Group
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720
(415) 642-6744
ucbvax!kridle

Measurements were made of file system throughput for various operations using a variety of currently available Winchester disks and controllers attached to both the native busses (SBI/CMI) and the UNIBUS on both VAX 780s and 750s. The tests were designed to measure the performance of single and dual drive subsystems operating in the 4.2BSD "fast file system" environment. Many of the results of the tests were initially counter intuitive and revealed several important aspects of the VAX implementations which were quite surprising to us.

The hardware used included two Fujitsu 2351 "Eagle" disk drives on each of two foreign vendor disk controllers and two DEC RA-81 disk drives on a DEC UDA-50 disk controller. The foreign vendor controllers were Emulex SC-750/780 and Systems Industries 9900 native bus interfaced controllers. The System Industries 9900 is heavily buffered, the Emulex controllers are not. The DEC UDA-50 controller is a UNIBUS interafaced, heavily buffered controller which is the first implementation of a new storage system architecture, DSA.

The tests were made up of a series of sequential reads or writes to files created in a newly initialized 4.2BSD file system. The block sizes of the reads and writes were varied as well as the "blocking factor" of the file system.

Tests were run first on single drives and then simultaneously on pairs of drives on the same controller. Results were reported in terms of aggregate throughput.

The most unexpected result of the testing was the sensitivity of VAX 780 throughput to the amount of buffering on the controller. This seems to be relatively independent of the location of the controller on the SBI or the UNIBUS. It was also observed that VAX 780 disk throughput seems to be a function of the amount of simultaneous CPU memory access. In other words, VAX 780s seem to be memory-bandwidth-bound, at least in the case of non-interleaved memory controllers.

Running the UNIX Kernel in User Mode

Michael Lutz
Computer Systems Consultant
103 Fox Chapel Road
Henrietta, NY 14467
(716) 359-2264
!rochester!ritcu!mjl

Michael Shon
GCA, Tropel Division
60 O'connor Road
Fairport, NY 14450
!rochester!ritcv!tropix!sys

In December, 1981, we began work on a project to port UNIX (Version 7) to an M68000-based EXORMACS system.

We already had UNIX running on a slower M68000 system, so most of the work consisted of adapting the kernel to a

different memory management unit (MMU). As it turned out, the differences had no effect on the memory organization of user programs, and we were able to run the development system's programs without recompilation or relinking.

The EXORMACS MMU is disabled when the processor is in supervisor state (the "natural" state for the UNIX kernel, which must be able to execute privileged instructions). The V7 kernel, however, implicitly assumes that the kernel address space is being mapped by the MMU.

We were faced with a dilemma: the V7 kernel needs access to privileged instructions, yet it also depends on memory mapping. We were unwilling to do major surgery on the kernel, so we had to find some way around this problem. The solution we finally adopted simply runs most of the kernel in user mode. The only exceptions to this are interrupt routines (which can neither do a process switch, nor modify the structure), the initial trap handler code, and some special assembly routines providing privileged operations to the the kernel. The MMU settings for the current user process and the V7 kernel are kept in two prototype data structures, and the support routines multiplex these onto the real MMU as needed. An additional set of C routines makes these prototypes look like the real MMU to the rest of the kernel.

Though our approach results in some performance degradation, this is limited to those portions of the kernel which are not time critical. What is more, the performance penalty is spread evenly over all the kernel operations, and there is no sharp spike when a process switch takes place. We consider the small performance penalty to be a modest price to pay for access to UNIX on the EXORMACS.

** M68000 and EXORMACS are trademark of Motorola, Inc.

Compilers and Languages 2

A General Purpose Programming Language with an Embedded Data Base Interface

Joel Isaacson

Sarris Computers
c/o New York Blood Center
310 E 67 St.
New York, NY 10021
(212) 229-7425
harpo!floyd!cmc12!presby!joel

QL is a high level, interpreted, general purpose language. It is meant to be used both by naive computer users and by seasoned programmers. QL has a data base interface which makes it ideal for complex queries of data bases. QL borrows much of its syntax from the C programming language, though the data types supported are richer.

This frees the programmer from much of the detail that writing applications in C code would entail, and produces programs which are typically 10-25% the size of equivalent C programs for data base queries.

The basic primitive data types of QL are:
1. Integers
2. Floating point
3. Strings (as a true data type, not as an array of characters)
4. Dates
5. Arrays (associative, nonhomogeneous combinations of the above types)
6. Forms (i.e. data base 'tuples')
7. Functions (can be recursive)
8. Data base keys.

A rich set of operators is provided, many with no C equivalent. In particular string manipulation and regular pattern matching are supported. A general type of array is supported. For example:

```
x ["abcd"] = 1.5;
x [3.5] [5] = 1;
```

will create an array *x* with the first element a floating point number with a string subscript and second element which is itself an array having a floating point subscript.

Mixed mode expressions of the basic types are allowed, i.e. if: *d* is of type date and *i* is of type integer (or float) then *d+i* is a date *i* days in the future (or past if *i* is negative) from day *d*.

Type declarations are absent, variables are assigned types dynamically as in APL. Type checking is performed by the interpreter. Run-time error messages are meant to be self-explanatory and contain the line number of the offending statement.

Turing: A New General Purpose Language Under Unix

J.R. Cordy and R.C. Holt

Computer Systems Research Group
University of Toronto
Toronto, Ontario, Canada

Tel: (416) 978-8715

Turing is a new general purpose programming language that is well suited for teaching programming. Turing can be thought of as a convenient, generalized, interactive version of Pascal. A Turing compiler has been developed at the University of Toronto. Turing is designed to support the development of reliable, efficient programs. It incorporates language features that decrease the cost of program development and that support formal program verification.

Turing is a Pascal-like language that incorporates almost all of Pascal's constructs. It alleviates many difficulties with Pascal; for example, Turing provides convenient string handling, it provides modules, its variant records (unions) are type safe, and it has dynamic parameters and arrays.

A Unix Tool Kit for Making Portable Compilers

Andrew S. Tanenbaum
Hans van Staveren
E.G. Keizer

Dept. of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
Telephone: 31 (20) 548 2410
decvax!mcvax!vu44!ast

The most fundamental of all software tools is the compiler. Without it, nearly all modern ideas about software engineering would be impractical. Surprisingly enough, compiler writing itself is something of a cottage industry, with each new compiler being handcrafted. The use of interchangeable parts, to allow a compiler to be brought up on a new machine by just changing a few tables is rarely encountered in practice. This paper describes a highly modular way of building compilers, allowing new languages and machines to be introduced without having to start all over each time. In addition, it describes some experience with the system, which has so far been used to produce 10 compilers (language-machine pairs).

The traditional way to produce compilers for L languages and M machines is wasteful: write L x M separate compilers from scratch. Our method consists of having L programs called front ends that translate from their respective source languages to a common intermediate language, which we have called EM (Encoding Machine). The intermediate code can then be passed through one or more optimizers, and translated to target assembly language by a table-driven program called a back end. Finally, the assembly language can be translated to machine language by a surprisingly machine-independent, table-driven universal assembler. With this scheme, adding a new language to the tool kit requires producing only a new front end, and adding a new machine requires writing two new tables, one for the back end and one for the assembler.

The idea of producing compilers using a common intermediate code (often called an UNCOL) is hardly new. What we have done is work out the details and actually make a practical implementation that runs on UNIX and produces high-quality compilers. At present, front ends for Pascal and C exist, with front ends for other languages under construction. Similarly, back end and assembler tables exist for the PDP-11, VAX, 8086, and 68000, with others in progress. The current version runs on the PDP-11 and VAX, with a 68000 version in the works.

UNIX Directions

Unix Style, or cat -v Considered Harmful

Rob Pike

Bell Labs 2C-521 Murray Hill NJ
(201) 582-7854
(dec!ucb)vax!research!rob

Unix is spreading rapidly throughout the commercial computing world. This spread is due largely to Unix's portability and practicality. Its original popularity in the academic computer science community, however, was due to its embodiment of a number of new and simple ideas, cleanly implemented.

The passage of time and programmers over every line of Unix source code has made the system more complicated to learn, to use, and to maintain, and much bigger -- the current VAX kernels are about a factor of 10 larger than the 5th Edition kernel, but certainly not a factor of 10 better. Most of that growth has not improved the system, but merely added to it.

This anecdotal presentation reviews some of the ideas responsible for Unix's early popularity, and shows by example that many of them have been forgotten, and that the "tools" idea has been superceded in practice by the creeping featurism that invades mature systems. But the tools philosphy is still vital, and more relevant than ever, so the talk closes with a few recent examples illustrating the power of assembling a program by interconnecting existing simple piece parts.

Everything You Wanted to Know About System V, and Then Some

Jim Balter

INTERACTIVE Systems Corporation
1212 Seventh Street
Santa Monica, CA 90401
(213) 450-8363
decvax!yale!ima!jim

In November 1982, AT&T pre-announced UNIX System V; in January 1983, AT&T announced it; in March, AT&T started selling System V manuals; and in April, AT&T started distributing the system itself.

This talk describes the differences between UNIX System III and UNIX System V, as gleaned from an examination of both documentation and code.

A description of some of the more subtle and/or amusing bugs is included, as is a list of supported configurations and devices.

UNIX System V and 4.1C BSD

John Chambers
Office of Academic Computing
449 Administration Building and Biostatistics
University of Texas Medical Branch
Galveston, TX 77550
(409) 761-1813
decvax!eagle!ut-ngp!jbc

John Quarterman
Computation Center
University of Texas at Austin
Austin, TX 78712
(512) 471-3241 x252
decvax!eagle!ut-ngp!jsq

A practical comparison of System V (the UNIX system Western Electric is currently licensing) and 4.1C BSD (the networking research UNIX developed for DARPA by the University of California at Berkeley), stemming from experience with both systems (on a VAX-11/750 and a VAX-11/780, respectively).

This paper compares the two systems in several areas, including: initial installation, booting, and configuration; languages, shells, typesetting, graphics, source code control, and data base; network and IPC support; terminal handler (fentl, ioctl, KMC-11 support) and other device drivers; games; operations, maintenance, and robustness; and portability.

Common features are for the most part left to the manuals, in order to concentrate on differences. This is a qualitative comparison, intended to serve only as a guide for further study (a bibliography is included). Some benchmark results are included, however.

** VAX and PDP are Trademarks of Digital Equipment Corporation

Berkeley UNIX after 4.2BSD
Where is it going and why do we want to get there?

Michael O'Dell

Department of Computer Science
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720
(415) 486-5583

There will be a brief discussion of the staff changes going on at UC Berkeley CSRG. The talk will then turn to the author's model of UNIX evolution and try to establish the current operating point within the model.

Future plans for improving the robustness, packaging, and maintainability of the system will be discussed, as will future directions for reducing system complexity, questionable features, and avoidable incompatability.

Networking

Local Network With Virtual Ports

Gary Gafke and Eric Bergan

Johns Hopkins University/Applied Physics Laboratory
Johns Hopkins Road
Laurel MD 20707
(301) 792-7800
brl-bmd!aplvax!gary

At the Johns Hopkins University Physics Laboratory, we have implemented a local area network connecting a variety of terminals and hosts. The network was developed at the Lab, and uses fiber optics and microprocessor controlled network interface units (NIUs). These NIUs relieve the hosts of most of the overhead associated with network protocols and also allow "dumb" terminals to be connected directly to the network.

Our network is now supporting several host computers including a VAX 11/780 running 4.1BSD Unix, a PDP-11/45 running PWB Unix, and a Zilog Z8000 system running Zenix. The network is also supporting approximately 12 terminals and assorted other peripherals. This network has been up and running in the user community since late February.

The PDP and Zilog computers are connected to the network using the traditional "milking machine" approach in which a separate physcial port on the host is required for each user (or other host) conversing with it over the network. However. a different approach was taken with the VAX. Since tty ports on a host are always at a premium, it was decided that for the VAX, a single port would be used to support many users. That is, we've developed a system of multiplexed virtual ports which map into a single physical port on the VAX.

Providing virtual ports on the VAX required software additions to Unix in the form of a pseudo (virtual device) driver and tty (virtual circuit) handler. This added approximately 15,000 bytes to the old version of UNIX which we had been running. However, the source code duplicates much of the new tty driver code to provide and insure independence. It can be trimmed down should space become a problem.

This paper describes the network configuration currently in place, the milking machine approach and the multiplexed approach. The changes that were made to Unix to provide virtual ports will also be described.

NETIX:
A UNIX-based network-using
operating system

Dr. A. Wambecq

Bell Telephone Manufacturing Company
Antwerp
(323) 237-1717

This talk will discuss the major design goals, and some implementation details, of the NETIX (1) network operating system. This system enables workstations, hosts and other equipment to work together in a concerted way. giving an integrated view of all services in the network. With this operating system, one can build systems for various application areas, like office systems, newspaper systems, or plant control.

NETIX is based on two building blocks:

- UNIX, an operating system originally designed for the PDP family of hardware, and now becoming a de facto operating system standard for 16 and 32 bit computer hardware.

- The local area network (LNA), including the intelligence that resides in its network interface units (NIU)(2).

Netix is designed to make the attached processors, the NIU's, and the LAN work together so that the user of the system is aware only of a single, networked machine, which unifies the files of all the processors, and makes processes run on whatever hardware is most appropriate.

This activity is transparent both to the user and to user programs. The result is uniform access to all resources (processing power, disk storage, etc.) by devices located throughout the network.

Each attached processor can work independently from the others, and each host processor can grant access by other machines to specific local resources.

(1) NETIX is a trademark of Bell Telephone Mfg. Co., ITT.
(2) NIU is a trademark of Ungermann-Bass, Inc.

EtherTIP - A Virtual Terminal Interface to Ethernet

Dick Foster

Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1
(403) 432-5640
alberta!dick

An EtherTip provides a group of terminals with a virtual terminal service. Each of these terminals can log on to any of the host systems connected to an Ethernet.

The current EtherTip hardware includes a SUN Workstation (MC68000 microprocessor + 256K RAM + Multibus + ...) without the normal keyboard and screen, but with the inclusion of a 3Com Multibus Ethernet interface, and an octal serial interface (provides 8 configurable RS-232 ports).

The network currently includes one VAX 11/780 (4.1bsd UNIX) and a PDP11/45 (2.8bsd UNIX), and 2 SUN workstations (4.2bsd UNIX). This will be expanded this summer with the addition of 3 VAX 11/780's with 4.2bsd UNIX and 4 more SUN workstations. 3Com's UNET is the networking software currently being used on the VAXen and PDP11.

The software for the EtherTip is loaded from a host through the Ethernet with the use of a load program (which could be located on a PROM in the EtherTip). Once the EtherTip system is loaded, a command mode (password protected) can be entered to configure each of the RS-232 ports to accommodate the particular terminal device it is to connect (e.g. set baud rates, parity, etc.). Parameters affecting the operation of the network software can also be altered (e.g. high water marks in buffers, retransmission timer characteristics, etc.). These parameters can be changed at any time by re-entering command mode.

The EtherTip uses the TCP/IP communication protocols, and the Telnet protocol for terminal support.

The EtherTip retains statistics on various parameters which can be checked as required to monitor performance (e.g. Number of ethernet received and transmitted, number of ethernet collisions, number of transport layer re-transmits, average round trip time, maximum round trip time, etc.).

Applications

A Data Base Frontend,
Driven by Tables Generated from a Data Dictionary

Edward Haenlin

New York Blood Center
310 East 67th Street
New York, NY 10021
(212) 570-3112
decvax!harpo!floyd!cmc12!nybcg!haenlin

"dbs" is a program which acts as a frontend for more than 100 New York Blood Center data bases. It enables a user at a terminal to enter, modify, display, print and delete data base contents.

"dbs" is table-driven, in the sense that it knows nothing about any specific data base until it reads a set of tables describing that data base. The tables are generated from a data dictionary by another program "pgen".

"pgen" makes reasonable assumptions about screen format, printing format, input sources, integrity constraints, default values, and availability of options and commands. Since these assumptions are incorporated in the tables, not in "dbs", one may alter them without programming -- and "dbs" appears to have been customized.

The data bases accessed by "dbs" take as their model a manual filing system. A data base, like a filing cabinet, contains a set of folders, which in turn contain forms, which consist of one or more items. A folder may contain more than one kind of form, and some forms may have multiple instances.

Data bases with related contents may be, and commonly are, linked in this system. "dbs" can provide simultaneous access to several interlinked data bases.

A Powerful Accounting Package for UNIX-Based Systems

Peter Wolfe & Allen Hustler

Human Computing Resources Corporation
10 St. Mary Street
Toronto, Ontario M4Y 1P9
(416) 922-1937

Although there have been many accounting packages introduced during the past decade, their capabilities have been constrained by the hardware and operating system limitations that prevailed at the time of their design. These limitations have included the cost and availability of main and secondary memory, the lack of a portable operating system, and the lack of a portable, widely implemented database model. As a result, these packages suffer from limited capabilities, rigid operational requirements, difficulties in adapting to the needs of specific users, auditing problems, and other deficiencies.

This paper describes the development of an accounting package designed specifically for the UNIX environment. Written entirely in C, and based on a portable interface to relational database systems, this package makes extensive use of UNIX tools to overcome the shortcomings of previous accounting packages.

UNIX Writer's Workbench

Charles R. Smith

Colorado State University
Department of English
Fort Collins, CO 80523
(303) 491-5310

Bell Laboratories' "UNIX Writer's Workbench" software is the first and still the only comprehensive series of programs for textual analysis. These programs help writers improve the quality of their work by offering analysis and criticism of on-line texts. Nearly two years prior to general release of these programs, a research exchange between Colorado State University and Bell Laboratories, Piscataway, New Jersey, permitted Colorado State University to test and adapt these remarkable programs for teaching composition. As a result, the fullest tests of the Workbench have taken place at CSU where the entire composition program, some 3000 students per year, is now taught with computer assistance and Bell's Workbench. This talk will focus on the CSU application, selected examples of output, and tests of effectiveness for improving editing and writing skill.

UNIX Mail

Where is Europe?

Jim McKie

Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
decvax!mcvax!jim

Some people are unclear as to where and what Europe is. This talk will give a brief history of UNIX in Europe, what is happening now, and future plans. Special emphasis will be placed on the features/bugs caused by having a group which has many differing nationalities and beaurocracies to deal with, and where the only common concept is UNIX; there are now 10 countries in Europe on the UUCP network, speaking 7 different languages. Europe also appears to have avoided many of the academic versus commercial user problems. The European UNIX Systems User Group has been a unifying force.

Unix and Electronic Mail:
Trials, Tribulations, and Proposals

Michael D. O'Dell

CSAM 50B/3238
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720
ucbvax!lbl-csam!mo

Unix and electronic mail have been friends for a long time, but the creation of large, complex computer networks has nearly been the undoing of this cherished friendship. This talk will explore some of the history of luminary electronic mail efforts based on Unix, examine the current state of exponential entropy, and offer some proposals for transforming the predictable into the desirable. Some of these proposals are guaranteed to be controversial.

Standards, Validation, and Portability

Developing a UNIX Validation Suite

Gary Fostel and Alison Naylor

North Carolina State University
P.O. Box 5972
Raliegh, NC 27650
decvax!duke!mcnc!ncsu!fostel

Indirect involvement with yet-another-UNIX-rehost project led us to investigate means of testing the "new" system. There are at least two ways in which this could be questioned:

First, there could be bugs which inhibited proper behavior, and

Second, with "all bugs fixed" there might remain legitimate, intentional differences between the new UNIX and ... indeed and what? BSD 4.1 UNIX? Or System III UNIX? Or 4.2 BSD, or System V? Or the forthcoming /usr/group Draft Standard?

There is a similar, although simpler problem with various implementations of supposedly standardized high-level languages. The Ada Validation Test Suite developed by Goodenough et al at Softech exemplifies the technique. Can this strategy be applied to Operating Systems such as UNIX? We believe the answer is "Yes, but ...".

Early Experiences Using UNIX
on the Gould SEL Concept Computers

Deborah L. Franke and Thomas R. Truscott

Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
(919) 541-6830(dlf), (919) 541-7005(trt)
mcnc!rti!dlf, mcnc!rti!trt

This talk describes RTI's experiences as an early user of the Gould 32/8750 computer running UNIX. The Gould 32/8750 is a 3.7 MIP-Whetstone computer that is potentially attractive as a high speed processor for computer aided design (CAD) applications. VLSI design tools and graphics tools written in C and Pascal have been ported to the 32/8750. Many of the programs were developed under Berkeley 4.1 BSD.

In addition, we will report our experiences using Ethernet between the Gould 32/8750 and a VAX 11/750. We feel that these experience should be valuable to others interested in porting existing UNIX programs or in writing portable programs in the future.

UNIX Version 7 compatibility
under System 3/5

Bob Scheulen

Microsoft Corporation
10700 Northup Way
Bellevue, WA 98004
(206) 828-8080
decvax!microsoft!bobs

Bell Unix System III and System V are rapidly replacing V7 as the preferred Unix version(s). Unfortunately, neither is upward compatible from V7. Today, most users of Unix-based systems are non-programmers who have neither the source code nor the expertise necessary to convert to System III.

This paper describes a System III release which provides binary and source level compatibility with both V7 and System III/V programs.

Items discussed include: ioctl peculiarities, system call support, and detection of V7 vs. SYS III/V programs.

Status Report from the UniForum Standards Committee

Heinz Lycklama

1212 Seventh Street
Santa Monica, CA 90401
(213) 450-8363

A report will be given on the current status of the Draft Standard for the system interface for Unix-like systems.

Netnews

I have reproduced below some of my network mail and a few "netnews" articles that I thought may be of interest to Australian UNIX users. I have deleted some of the less meaningful data generated by various mailers and news programs. No responsibility is taken for the accuracy (or lack thereof) of anything below.

From: mccallum@opus.UUCP
Newsgroups: net.unix-wizards
Subject: Re: panic: munhash (4.2BSD crash)
Date: Tue, 20-Dec-83 14:38:33 AESST

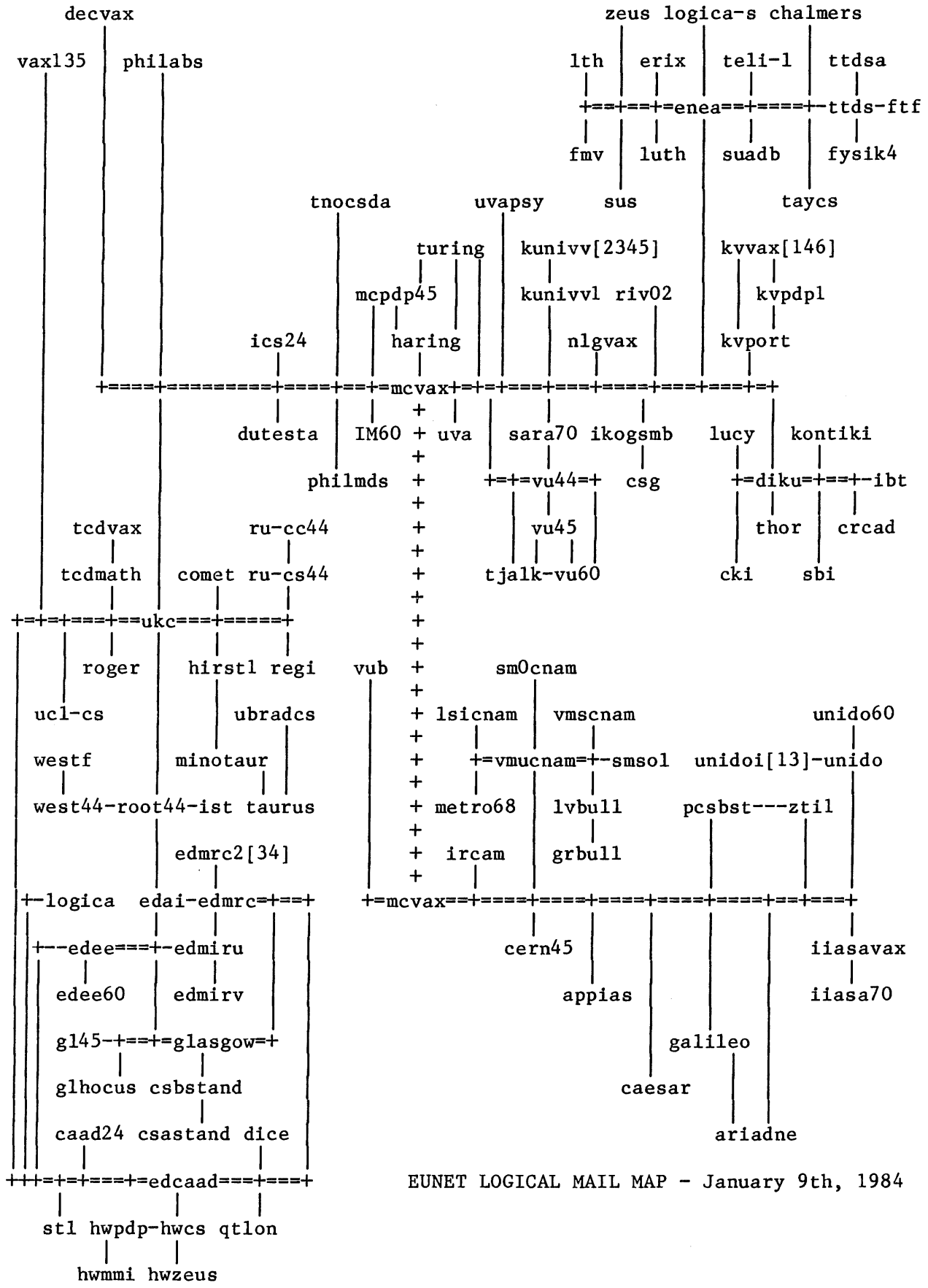
The bug fix for the panic: munhash in 4.2/4.1c has been posted.
The posted fix did not explain under what conditions the panic occurs.
The problem shows up when you have a LARGE file system and use a debugger
on program that resides in the part of the file system that makes the block
number field use all 20 bits. The fix is as follows:

> From: RWS%mit-xx@sri-unix.UUCP
> Newsgroups: net.unix-wizards
> Subject: sundry 4.2 bugs
> Date: Wed, 2-Nov-83 15:15:00 MST
>
> Despite claims to the contrary, the block number sign extension problem still exists. Berkeley put in a fix that should have worked, but a C compiler bug apparently keeps it from working. In /sys/sys/vm_mem.c in memall() the code
>
> ```
> swapdev : mount[c->c_mdev].m_dev, (daddr_t)(u_long)c->c_blkno
> ```
>
> should be changed to
>
> ```
> swapdev : mount[c->c_mdev].m_dev, c->c_blkno
> ```
>
> and in /sys/vax/vm_machdep.c in chgprot() the code
>
> ```
> munhash(mount[c->c_mdev].m_dev, (daddr_t)(u_long)c->c_blkno);
> ```
>
> should be changed to
>
> ```
> munhash(mount[c->c_mdev].m_dev, c->c_blkno);
> ```
>
> because the C compiler apparently incorrectly folds the (daddr_t) and (u_long) together and sign extends anyway. Simply taking out the (daddr_t)(u_long) works, although lint will probably complain about it.

From kre:munnari Tue Jan 17 21:29:11 1984
To: auugn:elecvax
Subject: European map

Complete UUCP configuration info is kept for continental Europe
by Piet Beertema (mcvax!piet) and for the UK by Mike Bayliss
(mcvax!ukc!mjb).

Here is a visual map of what there is.



EUNET LOGICAL MAIL MAP - January 9th, 1984

From: mayer@rochester.UUCP
Newsgroups: btl.unix,net.unix-wizards
Subject: Re: Use of encrypt (3C)
Date: Tue, 10-Jan-84 19:38:44 AESST

There is a bug in the encrypt/setkey stuff under 4.1c. The problem is that the "E table" is initialized only in the crypt routine. If encrypt/setkey are used independently of crypt the "E table" is left undefined. You can use the routines if you call "crypt" first, but the encryption will be done with a non standard table. To fix the problem, move the initialization "for" loop into "setkey". You will also have to move the declaration of the "e" and "E" arrays up above the "setkey" definition (otherwise they are undefined). The following stuff is a "diff" between the old and new versions.

-- Jim Mayer (rochester!mayer)

```
23a24,38
>  * The E bit-selection table.
>  */
> static        char    E[48];
> static        char    e[] = {
>       32, 1, 2, 3, 4, 5,
       ...
>       28,29,30,31,32, 1,
> };
>
> /*
104a120,125
>        *      Remember to initialize the E table first!
>        */
>       for(i=0;i<48;i++)
>               E[i] = e[i];
>
>       /*
144,158d164
<  * The E bit-selection table.
<  */
< static        char    E[48];
< static        char    e[] = {
<       32, 1, 2, 3, 4, 5,
       ...
<       28,29,30,31,32, 1,
< };
<
< /*
344,346d349
<
<       for(i=0;i<48;i++)
<               E[i] = e[i];
```

From: wiberg@chalmers.UUCP
Newsgroups: net.unix-wizards,net.bugs.4bsd
Subject: Cron dies - why! (Bug fix.)
Date: Thu, 26-Jan-84 14:43:46 AESST

It seems like some people out there have been bitten by the same
bug as we, so here's a diff list of our fix. It has worked as far
as we can see, which is only about a week.

```
1a2,17
>
> /*
>  *    cron.c - clock daemon.
>  *
>  *    Chalmers bug fix, 1984 01 23:
>  *    Test for list space (100 bytes ahead) in main loop of init()
>  *    was insufficient for long lines at certain places in crontab,
>  *    causing segmentation faults. It has been moved into each
>  *    nested loop and now has to think just a few bytes ahead at a
>  *    time, thus permitting arbitrarily long lines anywhere.
>  *    Also, an error test on the initial fork() has been added,
>  *    and harmless but redundant calls to free() before realloc()
>  *    have been removed.
>  *            CB & STW.
>  */
>
10d25
<
15a31
>
27a44
>       register int fret;
32,33c49,55
<      /*       setuid(1); */
<       if (fork())
---
>       /* setuid(1); */
>       if ((fret=fork()) < 0)          /* CB - 84 01 19 */
>       {
>               fprintf(stderr,"Cron: can't fork0);
>               exit(1);
>       }
>       else if(fret)
140,143c162,163
<       register i, c;
<       register char *cp;
<       register char *ocp;
<       register int n;
---
>       register i, c, n;
>       register char *cp, *ocp, *olist;
146,147c166
<       if (list) {
<               free(list);
---
>       if (list)
```

```
149c168
<       } else
---
>       else
154,162c173
< loop:
<       if(cp > list+listsize-100) {
<               char *olist;
<               listsize += LISTS;
<               olist = list;
<               free(list);
<               list = realloc(list, listsize);
<               cp = list + (cp - olist);
<       }
---
> loop:         /* List space test moved. STW - 84 01 23 */
172a184,188
>               if(cp >= list+listsize-3) {     /* STW - 84 01 23 */
>                       olist = list;
>                       list = realloc(list, listsize += LISTS);
>                       cp = list + (cp - olist);
>               }
195a212,216
>                       if(cp >= list+listsize-2) {     /* STW - 84 01 23 */
>                               olist = list;
>                               list = realloc(list, listsize += LISTS);
>                               cp = list + (cp - olist);
>                       }
218a240,244
>               if(cp >= list+listsize-3) {     /* STW - 84 01 23 */
>                       olist = list;
>                       list = realloc(list, listsize += LISTS);
>                       cp = list + (cp - olist);
>               }
229a256,260
>                       if(cp >= list+listsize-2) {     /* STW - 84 01 23 */
>                               olist = list;
>                               list = realloc(list, listsize += LISTS);
>                               cp = list + (cp - olist);
>                       }
```

Good luck!

Sven T. Wiberg @ Chalmers

From kre:munnari Fri Feb 3 18:46:15 1984
To: auugn:elecvax, decvax!aps
Subject: DEC are really starting to get there

Some mail I got from one of the 4.2bsd sites indicates that
DEC really are starting to know about unix, great work ...

Hi Robert. We are having machine check problems on our vax/750. About once a week (on average) we get a machine check 2 which causes UNIX to crash. Today our DEC service engineer brought out a UNIX patch which is supposed to get around the problem. The patch is dated 15/7/83 and is worded as follows.
****

The 11750 experiences TB parity errors due to a suspected noise problem with the L0003. The impact of the TB parity error varies with the rev level of the board and type of chips used and customer software operating system.

The effect of the parity errors is most critical when running UNIX or a modified version of VMS. UNIX in the current and previous releases from any vendor is not fault tolerant of vax-750 machine checks, simply displaying a console error message and "crashing/halting."

Following is a patch for the Berkeley version of UNIX to implement a simple error recovery scheme for TB PAR ERR machine checks. Its function is to "flush" the entire translation buffer by writing a "0" to the TBIA 1PR #39 (X) and prints a console message: "tbuf par: flushing and returning."

A. Locate the affected file: sys/machdep.c

B. Locate section of code to be changed (shown below):

```
#if VAX750
        case VAX_750: {
                register struct mc750frame *mci = (struct mc750frame *)cmcf;
                printf("va %x errpc %x mdr %x smr %x rdtimo %x tbgpar %x cacherr %x0,
                    mcf->mc5_va, mcf->mc5_errpc, mcf->mc5_mdr, mcf->mc5_svmode,
                    mcf->mc5_rdtimo, mcf->mc5_tbgpar, mcf->mc5_cacherr);
                printf("buserr %x mcesr %x pc %x psl %x mcsr %x0,
                    mcf->mc5_buserr, mcf->mc5_mcesr, mcf->mc5_pc, mcf->mc5_psl,
                    mfpr(MCSR));
                mtpr(MCESR, 0xf);

  ---------------->      <------------------ INSERT PATCH HERE

                break;
                }
#endif
```

C. Insert the code below just after the "mtpr(MESCR, 0xf);" line and before the "break;" line as flagged above:

```
        #define MC750_TBPAR 02
                if ((type&0xf) == MC750_TBPAR)
                {
```

```
        printf("tbuf par: flushing and returning0);
        mtpr(TBIA, 0);
        return;
        }
```

From: reidar@cucard.UUCP
Newsgroups: net.usenix
Subject: USENIX conferences
Date: Mon, 5-Mar-84 22:37:30 AESST

<Sorry, the first version of this was mangled for lack of this stuff>
Many people have noticed the friction which exists between USENIX and /usr/group, particularly at the conferences which have been jointly sponsored. It was evident in Washington that one of the major problems was a difference of emphasis between the two organizations; /usr/group gives much more importance to the vendor show, USENIX to the technical program. Another factor was /usr/group's evident intention to take over entirely at least the winter meetings.

It is the case that USENIX is not a co-sponsor of the conference in Dallas next January. It seems very likely that there will be a USENIX conference in January in Dallas at a different location from that of Uni-Forum (/usr/group's meeting). If there is to be a USENIX conference in Dallas a number of questions arise:

* Do most USENIX members want to attend a large convention such as UNIFORUM in Washington? or would a smaller somewhat less commercial (a scaled down vendor exhibit, for instance) meeting be preferable?

* Granting that the good old days are gone forever, is it possible, nevertheless, to hold a conference of a size and nature such that people who are serious UNIX users/hackers can talk to each other to their mutual benefit? Or is this sort of forum adequately provided by UUCP, USENET, and the various new UNIX-oriented publications?

* Does USENIX need to hold a conference more than once a year in the spring? Are there enough good presentations to warrant two full sessions of technical talks per year?

* Should USENIX hold only one meeting per year in June or July leaving the January meeting to /usr/group?

* More specifically, should USENIX hold a meeting in Dallas in Jan 1985?

--
Reidar Bornholdt

From decvax!mcvax!jim:mulga Mon Feb 6 22:15:21 1984
Subject: Re: need for information.
To: decvax!mulga!peteri:elecvax

Hi Peter. Sorry for any delay in a reply, your mail got caught in the great decvax snarl-up. However, I got both your messages OK.

Unfortunately I am a useless administrator, and am probably not the best person to ask these questions of. But I'll try.

We have, I believe, three real classes of membership, installation, vendor and individual. The annual fees are respectively 50, 40 and 17 pounds sterling. These fees were arrived at during a committee meeting in Nottingham in Sept. '81, no one has suggested they need changed. They were set to cover the costs of producing and mailing the newsletter, etc. In effect, there is no distinction between installation and vendor, except those that have, shall pay more! Individual members just get the newsletter and members fees for other services, but no voting rights (there are a surprising number of these outside Europe).

We get no financial help from anyone, although there will be a meeting between two of the committee and AT&T in Brussels next week. Perhaps we'll get something. The group does make a profit, however, by running bi-annual meetings. I can get a statement of accounts for you if you wish. There is also advertising in the newsletter (see below) and we have produced a catalogue of micro systems which run UNIX, and are preparing a catalogue of commercially available UNIX software. These go for 5 pounds to members, 10 pounds to non-members, plus vendors can pay for an advert beside their product entry.

Ah, the constitution. This is a problem. The last year has seen a desire to re-organise the EUUG due to pressure from non-UK members (when I moved here from Scotland I became the first committee member outside the UK!), so the constitution, which was never actually agreed on, has to be done again. What we have now is that the EUUG General Committee is composed of the chairmen of each country's NATIONAL group, and they should elect a small executive committee do do the day-to-day running (like a treasurer, newsletter editor, etc), since they can only really meet at the bi-annual meetings. National groups have almost complete autonomy, and can hold their own meetings, etc. Members join their national group, and, by right, then become members of the EUUG. The national group gives ~40% of the fees it collects to the EUUG. They can also ask the EUUG for financial assistance (like the Danish group did recently in order to finance an inaugural meeting to form the group). So the national group has a constitution, which is legally binding in whichever country we are talking about, but they are all different. I will send you a copy of an article which was in our newsletter last year about setting up national groups. It contains a skeleton of the Dutch group constitution (translated of course). We hope to sort out the real EUUG constitution at the next committee meeting, it's pretty messy at the moment.

One thing which made a lot of difference to the group was hiring a professional secretary to do all the grungy work like answering the phone, doing finances, mailing, reminding us lazy academics that we should do things, organising meetings, taking notes, minutes, etc. We have had that for about 18 months or so, and it has worked wonders.

I don't know how much it costs to print the newsletter, but I will find out. The phototypesetter is ours, and I run the stuff off myself (it is a Harris 7600, and we run ditroff, etc. There is a Versatec for previewing, a laser-printer is coming). I just send it to the EUUG Secretary in England, and she gets it printed professionally and mails it. I believe there are 500 printed, of which ~400 actually get mailed to members directly. The others are for inquiries, etc. We take advertising, rates are 100 pounds sterling per page, 50 pounds per half page, etc. I am not sure, but this should cover a lot of the expenses incurred by the newsletter.

Perhaps I should say after that that the group is still owned and run by academics, there is no desire to go in the direction of /usr/group (decidedly not!). We recently arranged a much closer reciprocal arrangement with USENIX, and the Software Tools Group, but there doesn't seem to be much in common between us and /usr/group. We just try to provide info and meetings that the members find useful. Meetings tend to be places where the beer is good, not places where businessmen feel at home.

<short pause as Jim gets a beer from the fridge>

I hope this has been of some help. If not, let me know and I will dig out someone to better answer the questions.

Sunny Australia? Sigh. There are large lumps of ice falling out the sky here at this precise moment.....

Cheers, Jim.

From mike:food23 Fri Feb 10 12:44:40 1984
To: netgurus:basservax
Subject: Computer jargon

From a recent 'The good whine', Telecom staff magazine:

VDU: A diseased sheep

Remote VDU: A diseased sheep in western NSW

Debug: De ting killed wid de pressure spray

Emulate: A tardy bird

Balanced Merge: Sex on a tight rope

Markov Chain: Used to tie up pavlovs dog

Microfiche: Plankton

Data Source: Makes fiche and chips taste better

Syntax: Royalties paid by brothel madam

Monostable: One horse accommodation.

From mh3bcl!research!wild!andrew:mulga Wed Feb 15 11:45:09 1984
Subject: edition 2
To: auugn:elecvax

Some random comments on Unix, 2nd edition, June 12 , 1972.

The list of authors has grown. It now includes Thompson, Ritchie, Ossana Morris, McIlroy, McMahon, Lorinda Cherry and Roberts.
The manual was done with 'ed' and 'roff' (as was the first).

1) cc only has one option (-c)!!

2) the bugs section for dsw reads:
'The name "dsw" is a carryover from the ancient past. Its etymology is amusing but the name is nonetheless ill-advised.'

3) find(1) takes file names or inode numbers and prints pwds of all matches.

4) pr(1) takes only 3 options (l==78 lines, c==current date, m==modify date)

5) ls(1) takes five (ltasd)

6) sort has no options!!

7) there is a cemt(2) system call to catch EMT traps.

8) there is a hog(2) system call (somewhat equivalent to nice(20))

9) catching interrupts is done by intr(2). The bugs reads
'It should be easier to resume after an interrupt but I don't know how to make it work.'

10) setting the modified date on a file is done by mdate(2)

11) sleep(2) sleeps for n/60 seconds. The bugs reads
'Due to the implementation the sleep interval is only accurate to 256/60 (4.26) seconds. Even then, the process is placed on a low priority queue and must be scheduled.'

12) qsort(3) uses its own comparison routine

13) the switch statement in C was supplied as a library function.
(C was just starting)

---

andrew

```
From: ado@elsie.UUCP
Newsgroups: net.bugs.4bsd
Subject: C compiler bug (better fix)
Date: Sat, 18-Feb-84 00:23:10 AESST

Subject: 4.?bsd C compiler error (better fix)
Index:  .../pcc/local2.c in 4.?BSD

Description:
        The C compiler generates incorrect code in some cases.
        A fix posted earlier fixed the problem while degrading code in some
        cases; this fix avoids code degradation.
Repeat-By:
        Use the command
                cc -S test.c
        where test.c contains:
                test()
                {
                        register struct {
                                short   i;
                                short   j;
                        } * sp;
                        while ((sp++)->i != 0);
                }
        and then look at the "test.s" file produced.  You'll see that the
        "while" loop generates this code:
                 L16:
                        tstw    (r11)+
                        jeql    L17
                        jbr     L16
        which only increments r11 by two (rather than four) each time through.
Fix:    This is the fix to the version distributed by Berkeley.

        ed - .../pcc/local2.c
        /ISPTR(p->in.left->in.type)/c
        #ifdef OLDVERSION
                                        if ( ISPTR(p->in.left->in.type) ) {
        #else
        /*
        ** We want to look for a pointer to a pointer, rather than a pointer.
        */
                                        if ( ISPTR(p->in.left->in.type) &&
                                           ISPTR(DECREF(p->in.left->in.type))) {
        #endif OLDVERSION
        .
        w
        q
--
UUCP:   decvax!harpo!seismo!rlgvax!cvl!elsie!ado
DDD:    (301) 496-5688
```

From: jmcg@decvax.UUCP
Newsgroups: net.mail
Subject: Re: Smart routing
Date: Sun, 19-Feb-84 09:05:46 AESST

As perpetrator of the path improver in use on decvax, sophomoric as it may be, I feel compelled to say a few words in its defense. Under the circumstances, it's probably just as well that I cannot remember who first suggested this method.

The path improver short circuits paths containing a uucp neighbor. For example, the path "siteA!siteB!siteC!user" is truncated to "siteC!user" if decvax has talked to siteC recently. Uucp (not just our version) keeps a stats file containing, for each site, the time of the most recent conversation or transfer.

The important qualities of this method are that it is local, automatic, fairly safe, and uniform. All of the information needed is available on the machine doing the routing. There is no database to be collected and maintained. Mail will not be mis-directed along paths that have fallen inactive nor redirected into loops. If everyone used it, it would work even better.

It does have failings. If it finds "vortex" in the path, then that piece of mail will go to LA rather than to the E-net. If someone sends mail on a long loop to probe a path or provoke a poll, it will get bounced at decvax rather than having the desired effect. It doesn't save us any money, just that of those sites we trim out of paths.

Jim McGinness
Digital Equipment Corp.
MKO2-1/H10
Merrimack, NH, 03054

decvax!jmcg
(603)844-5703

From: kre@munnari.SUN
Newsgroups: aus.news
Subject: News bug fix
Date: Sun, 4-Mar-84 01:04:06 AESST

The fix in net.announce for news/header.c applies to news source in Australia (all versions that I know about) - you should install it asap.

From: mark@cbosgd.UUCP
Newsgroups: net.announce
Subject: bug killers
Date: Sat, 25-Feb-84 18:08:16 AESST
Approved: mark@cbosgd.UUCP

I've been seeing lots of articles lately with "bug killers" or "wombat snacks" at the top. There is clearly a lot of misinformation going around about this, and I'm sure many of you are wondering what this is all about. Some people are even taking immune articles and making them vulnerable by adding these lines!

The problem is not completely fixed, but there are probably only 1 or 2 sites on Usenet that still have the bug. We hope to have it tracked down pretty soon and the last ones squashed. So the fear is real.

The problem is that, for articles whose body begins with white space (that is, you indent your first line by putting blanks or tabs on it), certain systems containing the bug may delete the first BUFSIZ characters of the body (not just the first line). For anyone who isn't sure if your system has fixed the bug, I'll enclose the fix.

For those of you worried about the articles you post, there are two things you can do:

The preferred way is to begin your article flush at the left margin, like this one. If you do that, there won't be a problem, and nobody has to be distracted by the extra line.

If you absolutely must indent that first line, then put an extra line in front of it that is not indented. (Only the first line matters, the remaining lines may be indented.) Thus:

```
<- bug killer
        This line is indented.
```

If you don't indent that first line, there is no problem:

This line is not indented.

The thing you absolutely do NOT want to do is include a bug killer that is indented:

```
        bug killer
```
This is NOT what you want to do, as this safe article becomes vulnerable!

Here is the fix:

Fix to header.c to fix batching problem chopping start of articles.

```
***************
*** 550,556
                /* Line too long - part read didn't fit into a newline */
                while ((c = getc(fp)) != '0 && c != EOF)
                        ;
!       } else
                *--tp = ' ';    /* clobber newline */
```

```
        while ((c = getc(fp)) == ' ' || c == '') {      /* for each cont line */

--- 550,558 -----
                /* Line too long - part read didn't fit into a newline */
                while ((c = getc(fp)) != '0 && c != EOF)
                        ;
!       } else if (tp == (cp+1))
!               return(cp);     /* Don't look for continuation of blank lines */
!       else
                *--tp = ' ';    /* clobber newline */

        while ((c = getc(fp)) == ' ' || c == '') {      /* for each cont line */
```

From: mogul%coyote@sri-unix.UUCP
Newsgroups: net.unix-wizards
Subject: fix for 4.2BSD kernel bug that trashes file systems
Date: Fri, 24-Feb-84 19:16:00 AESST

A few months ago, I sent a request to this list for help with a bug that was quietly trashing files and directories. I knew that the problem was a bad reference count on a file struct; I just wasn't sure how it got like that. Berkeley responded to me, with a fix that works fine. However, every few days I get a message from someone else who has the same problem, and since Berkeley has publicized this fix, I have to do so to keep my sanity.

My guess is that there are oodles of apparently bizarre problems that will be solved by installing this fix. Of course, I take no responsibility if it doesn't work for you!

------- Forwarded Message

From: karels%ucbmonet@Berkeley (Mike Karels)
Date: 13 Dec 1983 1606-PST (Tuesday)
To: Jeff Mogul <mogul@navajo>
Subject: Re: Serious 4.2 kernel bug causes files and directories to be mangled

You are right about the race in ino_close/closef, the problem can occur whenever the device close routine blocks for output to flush. We haven't seen the problem here (strangely), but it was discovered by Robert Elz. The changes that we have made follow; they have been running for a week or two on several machines without any problems, so I think there shouldn't be any problem. There are actually two changes; the first guarantees that closef will be done only once, even if interrupted, and the second catches interrupts in ino_close, which will then always return to closef. f_close can then be cleared exactly once. By the way, the ordering becomes more similar to that in 4.1.

Mike

Nov 18 10:06 1983 SCCS/s.kern_descrip.c: -r6.2 vs. -r6.3 Page 1

```
246,247d245
<       closef(fp);
<       /* WHAT IF u.u_error ? */
249a248,249
>       closef(fp);
>       /* WHAT IF u.u_error ? */
```

Nov 18 10:06 1983 SCCS/s.sys_inode.c: -r6.1 vs. -r6.2 Page 1

```
294c294
<       struct file *fp;
---
>       register struct file *fp;
296a297
>       register struct file *ffp;
309d309
```

```
<       fp->f_count = 0;                        /* XXX Should catch */
336,337c336,337
<       for (fp = file; fp < fileNFILE; fp++) {
<               if (fp->f_type == DTYPE_SOCKET)         /* XXX */
---
>       for (ffp = file; ffp < fileNFILE; ffp++) {
>               if (ffp == fp)
339c339,341
<               if (fp->f_count && (ip = (struct inode *)fp->f_data) &&
---
>               if (ffp->f_type == DTYPE_SOCKET)                /* XXX */
>                       continue;
>               if (ffp->f_count && (ip = (struct inode *)ffp->f_data) &&
352c354,363
<       (*cfunc)(dev, flag, fp);
---
>       if (setjmp(&u.u_qsave)) {
>               /*
>                * If device close routine is interrupted,
>                * must return so closef can clean up.
>                */
>               if (u.u_error == 0)
>                       u.u_error = EINTR;      /* ??? */
>               return;
>       }
>       (*cfunc)(dev, flag);
```

------- End of Forwarded Message

By the way, I strongly recommend, in sys/ufs_inode.c, in iput()
adding (before the first line of code):

```
        if (ip->i_count < 1)
                panic("iput: starting count < 1");
```

This will save you from similar sorts of trashing in the future (i.e.,
your system will crash but your files will not be randomly trashed.)
For those of you who remember a similar bug in the 4.1BSD mpx code,
this same panic would also have caught the problem.

-Jeff

From: presotto@rabbit.UUCP
Newsgroups: net.unix-wizards
Subject: 4.2 manuals
Date: Fri, 2-Mar-84 16:35:56 AEST

As announced at the UniForum Conference in Washington, D.C., USENIX is sponsoring the printing of 4.2BSD manuals. These manuals may be purchased only by USENIX members holding a 4.2BSD license agreement and will require each member to sign a simple agreement designating USENIX as their agent for manual duplication. There is no limit on the number of manuals a site may purchase, though we cannot promise there will be any printing runs other than this one. Consequently, it is highly recommended that sites consider this a one time event and order accordingly. Further, the larger the quantity of manuals printed, the lower the cost will be to all sites. This message has three purposes:

- to inform all interested parties of the imminent availability of 4.2 manuals,
- to publicize the expected format, and
- to collect responses from all those interested as to the quantity of manuals they expect to order.

The last item is most important as the total number of manuals printed will define the exact cost. Further, only those sites responding will receive a copy of the agreement necessary to purchase the manuals. Remember that these manuals will be sold only to USENIX members. If you are not a member of USENIX and wish to order manuals, you should join -- the cost to join will be easily recouped in the cost of the manuals purchased. (If you think this is a plug for USENIX, you're right.)

The information included below should answer most all questions about the manuals. If you have further questions regarding the manuals I will try and reply promptly if you send me mail at either of the addresses shown below.

Manual Descriptions
------ ------------

The 3 manuals which may be purchased are shown below; a detailed description of each volume's contents is given later. All manuals will be printed in a photo-reduced 6"x9" format with plastic binding which (unfortunately) does not permit local additions. Reference guides will have "bleed tabs" to ease the identification of manuals sections. The manual format and contents is fixed and no quantity of pleading will cause it to change (masters have already been created).

UNIX User's Manual (2 volumes)
        Volume 1, Reference Guide
        Volume 2, Supplementary Documents

UNIX Programmer's Manual (2 volumes)
        Volume 1, Reference Guide
        Volume 2, Supplementary Documents

UNIX System Manager's Manual (1 volume)

While some manuals are two separate volumes, one may only order complete manuals; i.e. one may NOT order a Volume 1 of the User's Manual without also ordering Volume 2.

The manuals are organized differently from the standard 4.2BSD manuals distributed by Berkeley for several reasons:

1. The quantity of material was too thick to permit the style of binding desired.
2. The reorganization permits grouping logically related information.
3. Most users will not have to shoulder the cost of printing material they will rarely use.
4. One may once again have manuals "small enough to fit in their briefcase" (though, of course, more of them).

Manual Contents
------ --------

UNIX User's Manual, Reference Guide

The following sections from Volume 1 of the original UPM: preface, introduction, table of contents, permuted index, section 1 (commands), section 6 (games), and section 7 (tables). Manual sections will have bleed tabs.

UNIX User's Manual, Supplementary Documents

The following documents from Volumes 2a, 2b, and 2c of the UPM:

7th Edition UNIX - Summary
The UNIX Time-Sharing System
UNIX for Beginners
A Tutorial Introduction to the UNIX Text Editor
Advanced Editing on UNIX
Edit: A Tutorial
An Introduction to Display Editing with Vi
Ex Reference Manual
An Introduction to the UNIX Shell
An Introduction to the C Shell
Learn - Computer Aided Instruction on UNIX
Mail Reference Manual
SED - A Non-interactive Text Editor
AWK - A Pattern Scanning and Processing Language
DC - An Interactive Desk Calculator
BC - An Arbitrary Precision Desk-Calculator Language
Typesetting Documents on the UNIX System
A Revised Version of -ms
Writing Papers with Nroff Using -me
-me Reference Manual
A System for Typesetting Mathematics
TBL - A Program to Format Tables
Some Applications of Inverted Indexes on the UNIX System
Refer - A Bibliography System
Writing Tools - The Style and Diction Programs
NROFF/TROFF User's Manual
A TROFF Tutorial
A Guide to the Dungeons of Doom

UNIX Programmer's Manual, Reference Guide

The following sections from Volume 1 of the original UPM: section 2 (system calls), section 3 (libraries), section 4 (devices), section 5 (file formats). All manual sections will have bleed tabs.

UNIX Programmer's Manual, Supplementary Documents

The following documents from Volumes 2a, 2b, and 2c of the UPM:

The C Programming Language - Reference Manual
The FRANZ LISP Manual
Berkeley Pascal User's Manual
Berkeley FP User's Manual
A Portable Fortran 77 Compiler
Introduction to the f77 I/O Library
Assembler Reference Manual
Lint, A C Program Checker
UNIX Programming
4.2BSD System Manual
A Tutorial Introduction to ADB
Make - A Program for Maintaining Computer Programs
YACC: Yet Another Compiler-Compiler
LEX - A Lexical Analyzer Generator
Rator - A Preprocessor for a Rational Fortran
The Programming Language EFL
The M4 Macro Processor
Screen Updating and Cursor Movement Optimization
An Introduction to the Source Code Control System
A Tour Through the Portable C Compiler

UNIX System Manager's Manual

Section 8 (maintenance commands) of the original UPM. The following documents from Volumes 2a, 2b, and 2c:

Installing and Operating 4.2BSD on the VAX
Building 4.2BSD UNIX System with Config
Fsck - The UNIX File System Check Program
4.2BSD Line Printer Spooler Manual
Sendmail - An Internetwork Mail Router
Sendmail Installation and Operation Guide
A Dial-Up Network of UNIX Systems
UUCP Implementation Description
UNIX Implementation
The UNIX I/O System
A Fast File System for UNIX
Disc Quotas in a UNIX Environment
4.2BSD Network Implementation Notes
On the Security of UNIX
Password Security: A Case History

Cost and Delivery

The cost of the manuals is still to be determined. Preliminary estimates indicate the cost breakdowns shown below. These costs will

vary according to the total number of each manual printed, the actual number of pages in each manual, and the printer selected. In addition to the charge for the printed material, each site will be responsible for paying shipping and handling charges. These charges also have yet to be determined. Assuming all goes well with the printing and verification of 4.2BSD license agreements, we hope the manuals will be available in late March or early April. We cannot give any more specific time of delivery.

User's Manual

Price per set (1000 copies printed) ~$22

Programmer's Manual

Price per set (1000 copies printed) ~$23

System Manager's Manual

Price per set (1000 copies printed) ~$16

This translates to ~$61 for a complete set of manuals (not including shipping), quite a bargain when one considers what other suppliers of manuals are charging!

Getting on the List

If you plan to order manuals when they become available, please fill out the short form below and return it via mail to:

ucbvax!manuals (uucp)
manuals@berkeley (ARPANET)

Alternatively (though not recommended), you may mail your form to me at

Sam Leffler
Lucasfilm, Ltd.
P.O. Box 2009
San Rafael, California 94912

To simplify my work, please be certain to include a Subject line in your message of the form given. Responses must be received within two weeks (March 14).

NOTE: This is NOT an order form! Do NOT send checks, purchase orders, etc. The purpose of responding at this time is to help us determine the quanties of manuals to print. All respondents will subsequently receive complete ordering information and a Manual Reproduction Authorization Form which must be signed and returned to USENIX along with proper license documentation before you can receive any manuals. To receive this form you must remember to include your US mailing address. Please do not indicate quantities based on expected costs (e.g. I'll take 10 of this if it costs $10, but only 5 if it costs $15), but rather accurate estimates based on your true expectations.

To: ucbvax!manuals

To: manuals@berkeley
Subject: USENIX manuals

Name:
Computer Mailing Address:
U.S. Mailing Address:
Phone Number:
USENIX Membership Number:

<number of> User's Manuals
<number of> Programmer's Manuals
<number of> System Manager's Manuals

From: mel@houxe.UUCP
Newsgroups: net.unix,net.unix-wizards,net.bugs.uucp
Subject: Another plea for area-code as uucp domains
Date: Sat, 21-Jan-84 22:37:58 AESST

At Uniforum yesterday, the "powers that be" in the uucp world announced that they were again undertaking to dictate "domains" for uucp. The problem was clearly presented: "In the next few years the number of uucp sites will increase to many thousand, and the present naming scheme is too chancy to prevent duplicate site names." The domain is a prefix or suffix to the site name to assure uniqueness by limiting the scope of the requirement for a unique name to just within the domain. Again, this was clearly presented: "The domain is to the site name, as the area-code it to a telephone number." (Their words, not mine.) My plea: "Please use the area-code for the uucp site name domain."

The telephone area-code as a uucp domain has the following advantages:
A. It is well known and understood,
B. It is easily defined: the area-code domain of a site is the same as the telephone number area-code of the main CPU of the site,
C. It is easily punctuated (My mail address is: (201)houxe!mel),
D. The area-code domain is sized by its population,
E. Every telephone book shows the domain locations and boundaries,
F. The area-code is limited to very few characters (just 3 in the US and Canada, 3 to 6 for other countries).

Thus, the area-code domain fits in fixed length fields in printed directories, business cards, forms, etc. It is easily parsed into whatever routing scheme one can think up. Any UNIX user can understand it, and how to use it. It has already proved itself in years of use by millions of real human beings.

Countrys and states are no good as domains as they are too big. Cities are too numerous. Company names are totally inappropriate; they aren't stable or well enough known (remember ABI? PARSEC? CSO?) (who is close to UNISOURCE? NUVATEK? ULTRIX?). Zip codes are too numerous. What else will do? Area-codes -or- an inadequate, inappropriate compromise complexity from some self-appointed central committee. PLEASE choose area-codes !!
Mel Haas , (201)houxe!mel

From kre:munnari Wed Mar 7 19:39:07 1984
To: auugn:elecvax
Subject: Re: Two returned messages

´:´s in addresses mean SUN here, if you want them to mean something else, they have to be hidden, I´ve been trying to find a way to mail to specific Berkeley hosts for a while now, without success. (ucbvax!person@host used to work, but not any more, something changed at their end).

You could try

ucbvax!g:usenix@decvax.uucp:mulga

it just might work (the @decvax.uucp bit might just hide the

Alternatively, Berkeley translate ´.´ to ´:´ on incoming mail, much as we do (but we only do it on incoming uucp mail) so maybe ´...!g.usenix´ would work.

A third possibility is

g:usenix@ucbvax.arpa:mulga

Another might be

g%usenix (with all the other stuff)

I really just don´t know.

We can´t make ´!´ take precedence over ´:´ as then we wouldn´t be able to send SUN mail to a host, then uucp it from there (which we want to do from time to time). But it doesn´t make sense to uucp to some host (from a SUN host that is) then send it by SUN from there, since SUN uses absolute addr´s, the "obvious" correct thing to do is send the mail direct by SUN, which would make the uucp part superfluous - but if the user specified it, he must have meant something, so ...

The real problem is that both Berknet (or whatever it is that Berkeley use to transport mail these days - ethernet I think) and SUN use ´:´ for addressing, and SUN uses it with host on the right, and Berknet & uucp have host on the left.

The address
a!b:c
is simply ambiguous. Someone has to simply define whether it means
(a!b):c or a!(b:c)

At mulga (and munnari), for the reasons above, I chose the former. The ´@´ has similar difficulties, but that one´s much easier for us to deal with. It has highest precedence, provided it is used in a form like
user@host.domain
(if there´s no ".domain" then the @ is simply treated as a ´:´,

and means SUN)

That's why
        g:usenix@ucbvax.arpa:mulga
might work (there's no need for "decvax!..." - in fact, that's likely
to screw things).

The second problem, two messages joined together, is something that
happens here with SUN between mulga & munnari, it seems to happen
randomly, with no apparent cause, and quite rarely. Just about
any two things might get joined together (provided they're both
in the queue at the same time & are being transmitted more or less together)

It doesn't happen often enough for me to worry about it, so
I'm not going to, if it continues after we start using ACSnet
full time between mulga & munnari, then I will dig into it
and see what's the cause.

Not a lot of help, am I?

Robert

From: dmr@research.UUCP
Newsgroups: net.mail.headers
Subject: addressing follies
Date: Tue, 28-Feb-84 07:41:37 AESST

Here, for your enjoyment, is the first line of the last 12 uucp letters
received by research from ucbvax. I'm especially fond of the
first one. Too bad it doesn't have a % to make it perfect.

```
From @Ucl-Cs.ARPA:@Caga.AC.UK:@Ucl-Cs.ARPA:lcp@Camsteve.AC.UK
From @SU-SCORE.ARPA:@SU-AI:MACKAY@WASHINGTON
From @SU-SCORE.ARPA:@SU-AI:greep@SU-DSN
From @SU-SCORE.ARPA:@SU-AI:phil@RICE
From @SU-SCORE.ARPA:@SU-AI:sdcarl!rusty@Berkeley
From @SU-SCORE.ARPA:DRF@SU-AI
From @SU-SCORE.ARPA:gwyn@brl-vld
From @SU-SCORE.ARPA:phil@RICE
From Ellis@YALE.ARPA
From fateman@ucbdali.Berkeley.ARPA
From kahn@UCLA-CS.ARPA
From lbl-csam!FURUTA@WASHINGTON.ARPA
```

From: wls@astrovax.UUCP
Newsgroups: net.bugs.4bsd,net.unix-wizards
Subject: 4.2 BSD bug in handling "tbuf par fault" on VAX 750
Date: Mon, 5-Mar-84 16:41:08 AEST

Index: /sys/vax/machdep.c 4.2BSD

Description:
The computer (a Vax 750) occasionally panics:
machine check 2: cp tbuf par fault
even though BSD 4.2 contains the patch to flush and return on tbuf parity errors.
The problem is that the test for the condition insists that bit 0 of the mcesr (prefetch reference bit) be zero, which need not be true.

Repeat-By:
Eventually the computer will panic as described above.

Fix:
Here are the diffs to /sys/vax/machdep.c. The line numbers of the new machdep.c may vary as there have been other fixes necessary.

```
*** machdep.c.ORIG      Tue Feb 28 11:27:04 1984
--- machdep.c   Mon Mar  5 15:10:41 1984
***************
*** 811,817
                    mcf->mc5_buserr, mcf->mc5_mcesr, mcf->mc5_pc, mcf->mc5_psl,
                    mfpr(MCSR));
                mtpr(MCESR, 0xf);
!               if ((mcf->mc5_mcesr&0xf) == MC750_TBPAR) {
                        printf("tbuf par: flushing and returning0);
                        mtpr(TBIA, 0);
                        return;

--- 822,828 -----
                    mcf->mc5_buserr, mcf->mc5_mcesr, mcf->mc5_pc, mcf->mc5_psl,
                    mfpr(MCSR));
                mtpr(MCESR, 0xf);
!               if ((mcf->mc5_mcesr&0xe) == MC750_TBPAR) {
                        printf("tbuf par: flushing and returning0);
                        mtpr(TBIA, 0);
                        return;
```

--
Bill Sebok Princeton University, Astrophysics
{allegra,akgua,burl,cbosgd,decvax,ihnp4,kpno,princeton,vax135}!astrovax!wls

Clippings this month come from "Whats New in Computing" December 1983, "The Gazette" February 1984 printed by Sydney University and "SIGPLAN Notices" Volume 18, No. 11, November 1983.

MULTI-USER DEVELOPMENT SYSTEM

In support of its NS16000 16/32-bit microprocessor family, National Semiconductor has introduced the SYS16 multi-user development system, a time-shared system that allows up to eight users access to up to 140 Mbytes of disc memory and to concurrently perform both emulation and software development. The SYS16 offers demand-paged virtual memory support, 32-bit registers, ALU, and internal data paths for rapid transmission over a 16-bit data bus linking the CPU to 32-bit floating point, memory management, and custom processor chips. Each user can independently address up to 16 Mbytes of memory, using virtual memory management to swap uniform 512 byte pages of programme or data directly between main and disc memories. The NS16000 CPU is able to support efficient high-level language programming with almost as little memory space as assembly languages and with nine dynamic addressing modes, is well suited to modular programming. The SYS16 comes with a GENIX operating system, fully supported by a C compiler (based on Berkeley's portable C compiler), NS16000 assembler, linker, libraries, utilities, loader, editor, and debugger. The GENIX time-shared, demand-paged system has protected address spaces, supporting from one to eight users and completely compatible with National's GENIX Cross-Software Package. A Pascal compiler is also available as an option. In its standard configuration, the SYS 16 processor module provides CPU, serial I/O, memory, and disc/tape controller boards installed in four of a total of six available connector slots. The remaining two slots accommodate optional memory boards. The CPU board contains an NS16032 CPU, an NS16201 timing control unit, an NS16202 interrupt control unit, an NS16081 floating point unit, an NS16082 memory management unit, diagnostic firmware, a parallel printer port, a GPIB IEEE488 port, an RS-232 port, and 256 Kbytes of RAM. The intelligent serial I/O board contains logic to support the eight RS-232 user ports which operate at up to 9600 baud and have FIFO buffers. The memory board contains 1 Mbyte of RAM with provisions for error checking and correction and access time of 400 ns. Additional memory boards may be added to the system, up to a total of 3.25 Mbytes. The disc/tape controller board contains electronics to control up to eight disc drives and streamer tape backup. The SYS16 - the disk/tape module houses an 8 in Winchester hard disc with a capacity of 20 Mbytes and comes with a ¼ in streamer tape (20 Mbyte capacity), as well as ready access for operating system and other software updates. Additional hard-disc memory may be added to the system by up to three disc-only modules of 40 Mbytes each. Hardware support is supplied in the form of a parallel printer interface compatible with industry-standards, and PROM programming with external unit. The ISE/16 NS16032 in-system emulator is available as an option. One terminal is supplied with the system as standard.

COMPUTER SCIENCE

Computer network has edge over US

COMPUTER scientists in the United States are looking upon a new UNIX networking system developed by the University of Sydney with almost as much envy as the New York Yacht Club viewed Australia II's victory in last year's Americas Cup.

The Americans would very much like to have a UNIX network with the flexibility, reliability and convenience of the Australian system, but they are already committed to an unwieldy system produced by the developers of UNIX, Bell Laboratories, and cannot easily change it.

Dr Bob Kummerfeld, a lecturer in the Basser Department of Computer Science, and Mr Piers Dick-Lauder, a senior programmer, are the originators of the Australian Computer Science Network. Set up in 1980 between four computers in Sydney, it now encompasses 72 different computers in institutions throughout Australia.

Dr Kummerfeld and Mr Dick-Lauder's system gives the Australian Computer Science Network a significant technological edge over its much larger American counterpart, USENET, which links together about 400 computers throughout the US.

The Australian and US systems are similar, in that they both use the well-proven and very popular UNIX Language C, developed by Bell Laboratories. Where they differ is that the Australian system does not use the accompanying UNIX networking package, known as Unix-Unix Copy Program (UUCP).

Dr Kummerfeld says that Bell Laboratories now admits that this program was something of an afterthought, and has serious short-comings.

'However, there are so many users of USENET now that it would be a mammoth task to change what's already been established. 'It would be a bit like changing the gauges of a large railway system', he said.

'Fortunately, when we were setting up our own computer science network three years ago, we rejected UUCP as being totally unsuited to our needs.'

The system which Dr Kummerfeld and Mr Dick-Lauder designed instead of UUCP has been so successful that the network is growing at the rate of 50 per cent per year.

In a country of long distances and slow, expensive communications, the network has helped to bring the computer science community much closer together. Also, a number of commercial users are interested in applying the system to their own companies, and in joining the network to keep up with new research developments.

Like a telex system, the computer network combines the advantages of telephones and letters. Communication can be instantaneous and interactive like telephones, and there can be a written record, as with letters. However, a computer-based messaging system is faster, cheaper and more versatile than telex and uses the same keyboard as researchers use to conduct other work.

'Electronic mail', as it is called, is so convenient that the computer scientists who now use it find it hard to imagine how they coped without it.

If a scientist in one institution wishes to contact several of his colleagues in different parts of the country to ask them for advice on a matter of teaching or research, all he needs to do is to type a message onto the keyboard in his office.

After adding a simple address, he can

Hanson - Smith, Ltd.
58 Martinka Drive
Shelton, Connecticut 06484
(203) 929 6133

20 July 1983

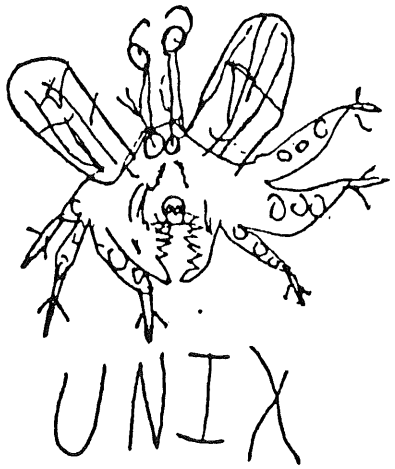
Richard L. Wexelblat, Editor
SIGPLAN Notices
ITT-ATC
1 Research Drive
Shelton, Connecticut 06484

Dear Mr. Wexelblat:

Some days ago, I had the following dialogue with my business partner's young son, Randy:

"Randy, please draw me a picture."
"All right, what should I draw?"
"Draw me a picture of a 'Unix,' okay?"
"A 'Unix,' huh? Let me see... that sounds like a bug."

The following was delivered:

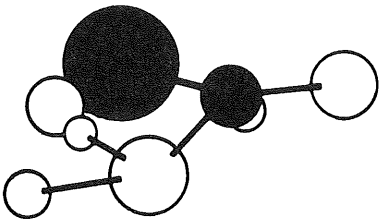


I suspected that others would appreciate this.

Very truly yours,

Trevor Russell Hanson

TRH:mas



HDE/emb

QUEENSLAND INSTITUTE OF TECHNOLOGY
GEORGE STREET, BRISBANE. Telex: 44699. Telegrams: Quintech, Brisbane.
G.P.O. BOX 2434, BRISBANE, QUEENSLAND, AUSTRALIA, 4001.

PHONE: (07) 223 2582

7th March 1984

Mr. P. Ivanov,
AUUGN Editor,
School of E.E. and C.S.,
University of New South Wales,
Box 1,
Post Office,
KENSINGTON. N.S.W. 2033

Dear Peter,

We are thinking of changing our 11/23 installation (includes ADV-11A A to D converter, for data logging in real time) from RSX11-M to UNIX. Besides data logging, we also use the 11/23 for straight computing, text editing, etc. and have FORTH and DECUS PASCAL as well as BASIC-11 and FORTRAN IV. Comments from others who have trod a similar path would be much appreciated.

Specific Queries

1. Is there a screen editor like EDT in UNIX?

2. Is there BASIC and FORTH for UNIX?

3. Is there likely to be an organisation like DECUS in the UNIX community to act as a resource centre for low-cost software? I think this will become important as UNIX spreads and non-expert people like me (I'm just a simple physicist) start to want user-friendly software.

Meanwhile, could you please send me whatever forms, etc. are needed to get our installation into the UNIX network, including cost of Newsletters, etc.

Yours sincerely,

Dan Ellis

(H.D. Ellis)
Senior Lecturer